Decision Trees

Nipun Batra and teaching staff

IIT Gandhinagar

August 13, 2025

Table of Contents

Introduction and Motivation

The need for interpretability

How to maintain trust in AI

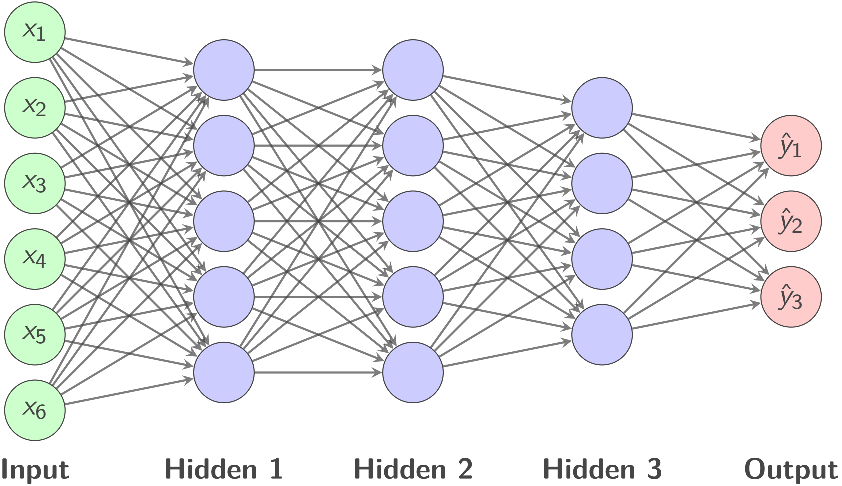
Beyond developing initial trust, however, creators of AI also must work to maintain that trust. Siau and Wang suggest seven ways of "developing continuous trust" beyond the initial phases of product development:

- Usability and reliability. AI "should be designed to operate easily and intuitively," Siau and Wang write. "There should be no unexpected downtime or crashes."
- Collaboration and communication. AI developers want to create systems that perform autonomously, without human involvement. Developers must focus on creating AI applications that smoothly and easily collaborate and communicate with humans.
- Sociability and bonding. Building social activities into AI applications is one way to strengthen trust. A robotic dog that can recognize its owner and show affection is one example, Siau and Wang write.
- Security and privacy protection. AI applications rely on large data sets, so ensuring privacy and security will be crucial to establishing trust in the applications.
- Interpretability. Just as transparency is instrumental in building initial trust, interpretability – or the ability for a machine to explain its conclusions or actions – will help sustain trust.

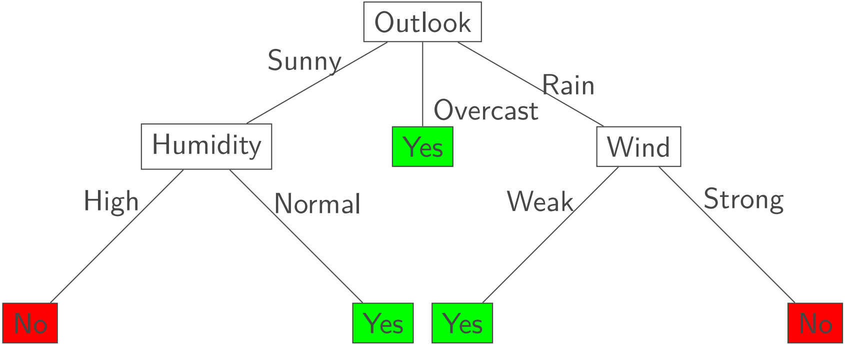
Training Data

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

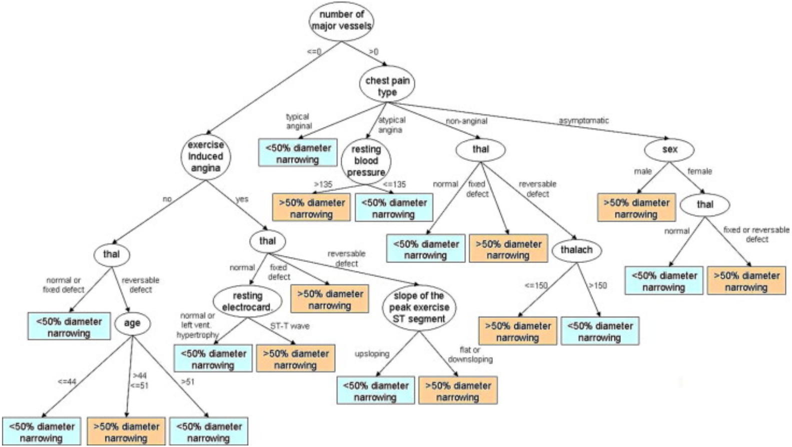
Learning a Complicated Neural Network



Learnt Decision Tree



Medical Diagnosis using Decision Trees



Source: Improving medical decision trees by combining relevant health-care criteria

Leo Breiman (1928-2005)

Leo Breiman 1928-2005

FOLLOW

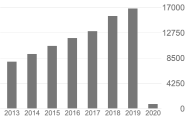
Professor of Statistics, UC Berkeley
Verified email at stat.berkeley.edu - Homepage

Data Analysis Statistics Machine Learning

TITLE	CITED BY	YEAR
Random forests L Breiman Machine learning 45 (1), 5-32	53816	2001
Classification and Regression Trees L Breiman, JH Friedman, RA Olshen, CJ Stone CRC Press, New York	~~43992~~ *	1999
Classification and regression trees L Breiman Chapman & Hall/CRC	43992 *	1984
Bagging predictors L Breiman Machine learning 24 (2), 123-140	22742	1996
Statistical Modeling: The Two Cutures L Breiman	2788 *	2003
Statistical modeling: The two cultures (with comments and a rejoinder by the author) L Breiman Statistical Science 16 (3), 199-231	~~2772~~	2001
Estimating optimal transformations for multiple regression and correlation	2096	1985

Cited by VIEW ALL

	All	Since 2015
Citations	142857	68736
h-index	51	33
i10-index	80	46



Leo Breiman: Revolutionary Contributions to ML

Key Points

Major Algorithmic Breakthroughs:

- **CART (1984)**: Classification and Regression Trees

Leo Breiman: Revolutionary Contributions to ML

Key Points

Major Algorithmic Breakthroughs:

- **CART (1984)**: Classification and Regression Trees
- **Bagging (1994)**: Bootstrap Aggregating

Leo Breiman: Revolutionary Contributions to ML

Key Points

Major Algorithmic Breakthroughs:

- **CART (1984)**: Classification and Regression Trees
- **Bagging (1994)**: Bootstrap Aggregating
- **Random Forests (2001)**: Ensemble of Decision Trees

Leo Breiman: Revolutionary Contributions to ML

Key Points

Major Algorithmic Breakthroughs:

- **CART (1984)**: Classification and Regression Trees
- **Bagging (1994)**: Bootstrap Aggregating
- **Random Forests (2001)**: Ensemble of Decision Trees
- **Two Cultures (2001)**: Data Modeling vs. Algorithmic Modeling

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time
  - Example: Given a sudoku solution, verify it's correct

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time
  - Example: Given a sudoku solution, verify it's correct
- **NP-Complete**: Hardest problems in NP

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time
  - Example: Given a sudoku solution, verify it's correct
- **NP-Complete**: Hardest problems in NP
  - Both in NP and at least as hard as any NP problem

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time
  - Example: Given a sudoku solution, verify it's correct
- **NP-Complete**: Hardest problems in NP
  - Both in NP and at least as hard as any NP problem
  - Example: Boolean satisfiability (SAT)

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time
  - Example: Given a sudoku solution, verify it's correct
- **NP-Complete**: Hardest problems in NP
  - Both in NP and at least as hard as any NP problem
  - Example: Boolean satisfiability (SAT)
- **NP-Hard**: At least as hard as NP-Complete problems

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time
  - Example: Given a sudoku solution, verify it's correct
- **NP-Complete**: Hardest problems in NP
  - Both in NP and at least as hard as any NP problem
  - Example: Boolean satisfiability (SAT)
- **NP-Hard**: At least as hard as NP-Complete problems
  - May not be in NP (solutions might not be verifiable quickly)

Computational Complexity Classes: A Quick Primer

Definition: Key Complexity Classes

- **P**: Problems solvable in polynomial time
  - Example: Sorting n numbers in $O(n \log n)$ time
- **NP**: Problems where solutions can be *verified* in polynomial time
  - Example: Given a sudoku solution, verify it's correct
- **NP-Complete**: Hardest problems in NP
  - Both in NP and at least as hard as any NP problem
  - Example: Boolean satisfiability (SAT)
- **NP-Hard**: At least as hard as NP-Complete problems
  - May not be in NP (solutions might not be verifiable quickly)
  - Example: Optimization versions of NP-Complete problems

Finding the Optimal Decision Tree

Volume 5, number 1 INFORMATION PROCESSING LETTERS May 1976

CONSTRUCTING OPTIMAL BINARY DECISION TREES IS NP-COMPLETE*

Laurent HYAFIL
IRIA – Laboria, 78150 Rocquencourt, France

and

Ronald L. RIVEST
Dept. of Electrical Engineering and Computer Science, M.I.T., Cambridge, Massachusetts 02139, USA

Received 7 November 1975, revised version received 26 January 1976

Binary decision trees, computational complexity, NP-complete

The Problem: Given training data, find the decision tree with the highest accuracy

Optimal Decision Trees are NP-Complete

Important: Computational Complexity

Finding optimal decision tree is NP-Complete

- **Verification**: Given a tree, check its accuracy quickly ✓

Optimal Decision Trees are NP-Complete

Important: Computational Complexity

Finding optimal decision tree is NP-Complete

- **Verification**: Given a tree, check its accuracy quickly ✓
- **Construction**: Exponentially many trees to check ✗

Optimal Decision Trees are NP-Complete

Important: Computational Complexity

Finding optimal decision tree is NP-Complete

- **Verification**: Given a tree, check its accuracy quickly ✓
- **Construction**: Exponentially many trees to check ✗

Example: What This Means

- No efficient algorithm exists (unless P $=$ NP)

Optimal Decision Trees are NP-Complete

Important: Computational Complexity

Finding optimal decision tree is NP-Complete

- **Verification**: Given a tree, check its accuracy quickly ✓
- **Construction**: Exponentially many trees to check ×

Example: What This Means

- No efficient algorithm exists (unless P = NP)
- Must use heuristics like greedy algorithms

Optimal Decision Trees are NP-Complete

Important: Computational Complexity

Finding optimal decision tree is NP-Complete

- **Verification**: Given a tree, check its accuracy quickly ✓
- **Construction**: Exponentially many trees to check ×

Example: What This Means

- No efficient algorithm exists (unless P = NP)
- Must use heuristics like greedy algorithms
- ID3, C4.5, CART use greedy approaches

Optimal Decision Trees are NP-Complete

Important: Computational Complexity

Finding optimal decision tree is NP-Complete

- **Verification**: Given a tree, check its accuracy quickly ✓
- **Construction**: Exponentially many trees to check ×

Example: What This Means

- No efficient algorithm exists (unless P = NP)
- Must use heuristics like greedy algorithms
- ID3, C4.5, CART use greedy approaches
- Good solutions, but no optimality guarantee

Greedy Algorithm

Core idea: At each level, choose an attribute that gives **biggest estimated** performance gain!

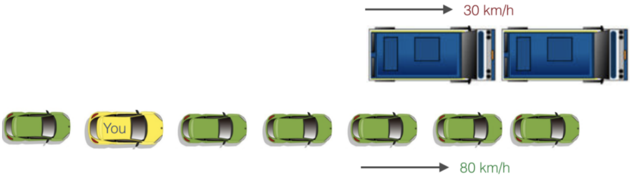


Image source: analyticsvidhya

Greedy $\neq$ Optimal

Discrete Input, Discrete Output

Towards biggest estimated performance gain

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- For examples, we have 9 Yes, 5 No

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- For examples, we have 9 Yes, 5 No
- Would it be trivial if we had 14 Yes or 14 No?

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- For examples, we have 9 Yes, 5 No
- Would it be trivial if we had 14 Yes or 14 No?
- Yes!

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- For examples, we have 9 Yes, 5 No
- Would it be trivial if we had 14 Yes or 14 No?
- Yes!
- Key insight: Problem is "easier" when there is less disagreement

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- For examples, we have 9 Yes, 5 No
- Would it be trivial if we had 14 Yes or 14 No?
- Yes!
- Key insight: Problem is "easier" when there is less disagreement
- Need some statistical measure of "disagreement"

Claude Shannon (1948): The Birth of Information Theory

Definition: The Big Idea

Information is inversely related to probability. **Rare events are more informative!**

Claude Shannon (1948): The Birth of Information Theory

Definition: The Big Idea

Information is inversely related to probability. **Rare events are more informative!**

Think about it: Which headline tells you more?

Claude Shannon (1948): The Birth of Information Theory

Definition: The Big Idea

Information is inversely related to probability. **Rare events are more informative!**

Think about it: Which headline tells you more?

- "The sun rose this morning"

Claude Shannon (1948): The Birth of Information Theory

Definition: The Big Idea

Information is inversely related to probability. **Rare events are more informative!**

Think about it: Which headline tells you more?

- "The sun rose this morning"
- "It snowed in Gandhinagar in July"

Claude Shannon (1948): The Birth of Information Theory

Definition: The Big Idea

Information is inversely related to probability. **Rare events are more informative!**

Think about it: Which headline tells you more?

- "The sun rose this morning"
- "It snowed in Gandhinagar in July"

The second one! Because it's **unexpected**.
Shannon's insight: The amount of information in an event should be inversely proportional to its probability.

Measuring Surprise: Step by Step

Shannon's Information Formula:

$$I(x) = -\log_2 p(x)$$

Measuring Surprise: Step by Step

Shannon's Information Formula:

$$I(x) = -\log_2 p(x)$$

Why the negative log?

- Probabilities are between 0 and 1

Measuring Surprise: Step by Step

Shannon's Information Formula:

$$I(x) = -\log_2 p(x)$$

Why the negative log?

- Probabilities are between 0 and 1
- $\log_2$ of values < 1 gives negative numbers

Measuring Surprise: Step by Step

Shannon's Information Formula:

$$I(x) = -\log_2 p(x)$$

Why the negative log?

- Probabilities are between 0 and 1
- $\log_2$ of values < 1 gives negative numbers
- We want information to be positive

Measuring Surprise: Step by Step

Shannon's Information Formula:

$$I(x) = -\log_2 p(x)$$

Why the negative log?

- Probabilities are between 0 and 1
- $\log_2$ of values < 1 gives negative numbers
- We want information to be positive
- Hence the negative sign!

Measuring Surprise: Step by Step

Shannon's Information Formula:

$$I(x) = -\log_2 p(x)$$

Why the negative log?

- Probabilities are between 0 and 1
- $\log_2$ of values < 1 gives negative numbers
- We want information to be positive
- Hence the negative sign!

Why base 2? So information is measured in **bits**.

Calculating Surprise: Detailed Examples

Example 1: Summer weather in Phoenix

- Sunny day: $p = 0.9$

Calculating Surprise: Detailed Examples

Example 1: Summer weather in Phoenix

- Sunny day: $p = 0.9$
- Information: $I = -\log_2(0.9) = -(-0.152) = 0.152$ bits

Calculating Surprise: Detailed Examples

Example 1: Summer weather in Phoenix

- Sunny day: $p = 0.9$
- Information: $I = -\log_2(0.9) = -(-0.152) = 0.152$ bits
- **Low surprise** - we expect sunny weather

Calculating Surprise: Detailed Examples

Example 1: Summer weather in Phoenix

- Sunny day: $p = 0.9$
- Information: $I = -\log_2(0.9) = -(-0.152) = 0.152$ bits
- **Low surprise** - we expect sunny weather

Example 2: Snow in Phoenix in July

- Probability: $p = 0.0001$ (extremely rare!)

Calculating Surprise: Detailed Examples

Example 1: Summer weather in Phoenix

- Sunny day: $p = 0.9$
- Information: $I = -\log_2(0.9) = -(-0.152) = 0.152$ bits
- **Low surprise** - we expect sunny weather

Example 2: Snow in Phoenix in July

- Probability: $p = 0.0001$ (extremely rare!)
- Information: $I = -\log_2(0.0001) = -(-13.29) = 13.29$ bits

Calculating Surprise: Detailed Examples

Example 1: Summer weather in Phoenix

- Sunny day: $p = 0.9$
- Information: $I = -\log_2(0.9) = -(-0.152) = 0.152$ bits
- **Low surprise** - we expect sunny weather

Example 2: Snow in Phoenix in July

- Probability: $p = 0.0001$ (extremely rare!)
- Information: $I = -\log_2(0.0001) = -(-13.29) = 13.29$ bits
- **High surprise** - this would be shocking news!

Calculating Surprise: Detailed Examples

Example 1: Summer weather in Phoenix

- Sunny day: $p = 0.9$
- Information: $I = -\log_2(0.9) = -(-0.152) = 0.152$ bits
- **Low surprise** - we expect sunny weather

Example 2: Snow in Phoenix in July

- Probability: $p = 0.0001$ (extremely rare!)
- Information: $I = -\log_2(0.0001) = -(-13.29) = 13.29$ bits
- **High surprise** - this would be shocking news!

Notice: Rare events carry $\sim 90\times$ more information!

From Single Events to Distributions

Question: What if we have multiple possible outcomes?

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$
- Cloudy: $p = 0.3$

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$
- Cloudy: $p = 0.3$
- Sunny: $p = 0.15$

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$
- Cloudy: $p = 0.3$
- Sunny: $p = 0.15$
- Snowy: $p = 0.05$

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$
- Cloudy: $p = 0.3$
- Sunny: $p = 0.15$
- Snowy: $p = 0.05$

Problem: Each day gives different amounts of information!

- If it's rainy: $I = -\log_2(0.5) = 1.0$ bit

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$
- Cloudy: $p = 0.3$
- Sunny: $p = 0.15$
- Snowy: $p = 0.05$

Problem: Each day gives different amounts of information!

- If it's rainy: $I = -\log_2(0.5) = 1.0$ bit
- If it's sunny: $I = -\log_2(0.15) = 2.74$ bits

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$
- Cloudy: $p = 0.3$
- Sunny: $p = 0.15$
- Snowy: $p = 0.05$

Problem: Each day gives different amounts of information!

- If it's rainy: $I = -\log_2(0.5) = 1.0$ bit
- If it's sunny: $I = -\log_2(0.15) = 2.74$ bits
- If it's snowy: $I = -\log_2(0.05) = 4.32$ bits

From Single Events to Distributions

Question: What if we have multiple possible outcomes?
Example: Weather in Delhi (4 possibilities)

- Rainy: $p = 0.5$
- Cloudy: $p = 0.3$
- Sunny: $p = 0.15$
- Snowy: $p = 0.05$

Problem: Each day gives different amounts of information!

- If it's rainy: $I = -\log_2(0.5) = 1.0$ bit
- If it's sunny: $I = -\log_2(0.15) = 2.74$ bits
- If it's snowy: $I = -\log_2(0.05) = 4.32$ bits

Solution: Take the **expected** (average) information!

Entropy: Expected Information

Definition: Entropy Formula

$$H(X) = \mathbb{E}[I(X)] = -\sum_i p(x_i) \log_2 p(x_i)$$

Entropy = Expected amount of information per observation

Entropy: Expected Information

Definition: Entropy Formula

$$H(X) = \mathbb{E}[I(X)] = -\sum_i p(x_i) \log_2 p(x_i)$$

Entropy = Expected amount of information per observation

Delhi weather calculation:

Entropy: Expected Information

Definition: Entropy Formula

$$H(X) = \mathbb{E}[I(X)] = -\sum_i p(x_i) \log_2 p(x_i)$$

Entropy = Expected amount of information per observation

Delhi weather calculation:

$$\begin{aligned} H = &-p(\text{rain}) \log_2 p(\text{rain}) - p(\text{cloudy}) \log_2 p(\text{cloudy}) \\ &- p(\text{sunny}) \log_2 p(\text{sunny}) - p(\text{snow}) \log_2 p(\text{snow}) \end{aligned}$$

Entropy: Expected Information

Definition: Entropy Formula

$$H(X) = \mathbb{E}[I(X)] = -\sum_i p(x_i) \log_2 p(x_i)$$

Entropy = Expected amount of information per observation

Delhi weather calculation:

$$\begin{aligned} H &= -p(\text{rain}) \log_2 p(\text{rain}) - p(\text{cloudy}) \log_2 p(\text{cloudy}) \\ &\quad - p(\text{sunny}) \log_2 p(\text{sunny}) - p(\text{snow}) \log_2 p(\text{snow}) \\ &= -0.5 \log_2(0.5) - 0.3 \log_2(0.3) - 0.15 \log_2(0.15) - 0.05 \log_2(0.05) \end{aligned}$$

Entropy: Expected Information

Definition: Entropy Formula

$$H(X) = \mathbb{E}[I(X)] = -\sum_i p(x_i) \log_2 p(x_i)$$

Entropy = Expected amount of information per observation

Delhi weather calculation:

$$\begin{aligned} H &= -p(\text{rain}) \log_2 p(\text{rain}) - p(\text{cloudy}) \log_2 p(\text{cloudy}) \\ &\quad - p(\text{sunny}) \log_2 p(\text{sunny}) - p(\text{snow}) \log_2 p(\text{snow}) \\ &= -0.5 \log_2(0.5) - 0.3 \log_2(0.3) - 0.15 \log_2(0.15) - 0.05 \log_2(0.05) \\ &= 0.5(1.0) + 0.3(1.74) + 0.15(2.74) + 0.05(4.32) \end{aligned}$$

Entropy: Expected Information

Definition: Entropy Formula

$$H(X) = \mathbb{E}[I(X)] = -\sum_i p(x_i) \log_2 p(x_i)$$

Entropy = Expected amount of information per observation

Delhi weather calculation:

$$\begin{aligned}
H &= -p(\text{rain}) \log_2 p(\text{rain}) - p(\text{cloudy}) \log_2 p(\text{cloudy}) \\
&\quad - p(\text{sunny}) \log_2 p(\text{sunny}) - p(\text{snow}) \log_2 p(\text{snow}) \\
&= -0.5 \log_2(0.5) - 0.3 \log_2(0.3) - 0.15 \log_2(0.15) - 0.05 \log_2(0.05) \\
&= 0.5(1.0) + 0.3(1.74) + 0.15(2.74) + 0.05(4.32) \\
&= 0.5 + 0.52 + 0.41 + 0.22 = \mathbf{1.65} \text{ bits}
\end{aligned}$$

Entropy Intuition: Extreme Cases

Case 1: Completely predictable

- Desert: Always sunny ($p = 1.0$)

Entropy Intuition: Extreme Cases

Case 1: Completely predictable

- Desert: Always sunny ($p = 1.0$)
- $H = -1.0 \log_2(1.0) = -1.0 \times 0 = \mathbf{0}$ bits

Entropy Intuition: Extreme Cases

Case 1: Completely predictable

- Desert: Always sunny ($p = 1.0$)
- $H = -1.0 \log_2(1.0) = -1.0 \times 0 = \mathbf{0}$ bits
- **Zero entropy** = No surprise = Completely predictable

Entropy Intuition: Extreme Cases

Case 1: Completely predictable

- Desert: Always sunny ($p = 1.0$)
- $H = -1.0 \log_2(1.0) = -1.0 \times 0 = \mathbf{0}$ bits
- **Zero entropy** = No surprise = Completely predictable

Case 2: Maximum uncertainty

- Fair coin: Heads/Tails equally likely ($p = 0.5$ each)

Entropy Intuition: Extreme Cases

Case 1: Completely predictable

- Desert: Always sunny ($p = 1.0$)
- $H = -1.0 \log_2(1.0) = -1.0 \times 0 = \mathbf{0}$ bits
- **Zero entropy** = No surprise = Completely predictable

Case 2: Maximum uncertainty

- Fair coin: Heads/Tails equally likely ($p = 0.5$ each)
- $H = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 0.5(1) + 0.5(1) = \mathbf{1.0}$ bit

Entropy Intuition: Extreme Cases

Case 1: Completely predictable

- Desert: Always sunny ($p = 1.0$)
- $H = -1.0 \log_2(1.0) = -1.0 \times 0 = \mathbf{0}$ bits
- **Zero entropy** = No surprise = Completely predictable

Case 2: Maximum uncertainty

- Fair coin: Heads/Tails equally likely ($p = 0.5$ each)
- $H = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 0.5(1) + 0.5(1) = \mathbf{1.0}$ bit
- **Maximum entropy** = Maximum surprise = Completely unpredictable

Entropy Intuition: Extreme Cases

Case 1: Completely predictable

- Desert: Always sunny ($p = 1.0$)
- $H = -1.0 \log_2(1.0) = -1.0 \times 0 = \mathbf{0}$ bits
- **Zero entropy** = No surprise = Completely predictable

Case 2: Maximum uncertainty

- Fair coin: Heads/Tails equally likely ($p = 0.5$ each)
- $H = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 0.5(1) + 0.5(1) = \mathbf{1.0}$ bit
- **Maximum entropy** = Maximum surprise = Completely unpredictable

Key insight: Entropy ranges from 0 (certain) to $\log_2(n)$ (uniform over n outcomes)

Entropy in Decision Trees: The Connection

Why do we care about entropy in ML?

Entropy in Decision Trees: The Connection

Why do we care about entropy in ML?

Example: Decision Tree Goal

We want to split data into **pure** subsets where we can make confident predictions.

Entropy in Decision Trees: The Connection

Why do we care about entropy in ML?

Example: Decision Tree Goal

We want to split data into **pure** subsets where we can make confident predictions.

- **Pure node**: All examples same class → Low entropy → Good split

Entropy in Decision Trees: The Connection

Why do we care about entropy in ML?

Example: Decision Tree Goal

We want to split data into **pure** subsets where we can make confident predictions.

- **Pure node**: All examples same class $\rightarrow$ Low entropy $\rightarrow$ Good split
- **Mixed node**: Examples from different classes $\rightarrow$ High entropy $\rightarrow$ Bad split

Entropy in Decision Trees: The Connection

Why do we care about entropy in ML?

Example: Decision Tree Goal

We want to split data into **pure** subsets where we can make confident predictions.

- **Pure node**: All examples same class $\rightarrow$ Low entropy $\rightarrow$ Good split
- **Mixed node**: Examples from different classes $\rightarrow$ High entropy $\rightarrow$ Bad split

Strategy: Choose splits that **reduce entropy** the most! This is exactly what **Information Gain** measures.

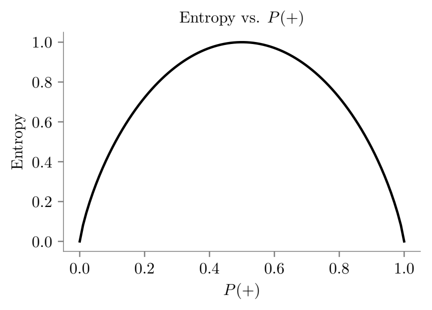
Entropy

Statistical measure to characterize the (im)purity of examples

Entropy

Statistical measure to characterize the (im)purity of examples
$H(X) = -\sum_{i=1}^{k} p(x_i) \log_2 p(x_i)$

Notebook: entropy.html



Towards biggest estimated performance gain

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- Can we use Outlook as the root node?

Towards biggest estimated performance gain

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- Can we use Outlook as the root node?
- When Outlook is overcast, we always Play and thus no "disagreement"

Information Gain

Reduction in entropy by partitioning examples (S) on attribute A

$$\mathrm{Gain}(S, A) \equiv \mathrm{Entropy}(S) - \sum_{v \in \mathrm{Values}(A)} \frac{|S_v|}{|S|} \mathrm{Entropy}(S_v)$$

Pop Quiz #1

Answer this!

What does entropy measure in the context of decision trees?

A) The depth of the tree
B) The impurity or “disagreement” in a set of examples
C) The number of features in the dataset
D) The accuracy of the tree

Pop Quiz #1

Answer this!

What does entropy measure in the context of decision trees?

A) The depth of the tree

B) The impurity or "disagreement" in a set of examples

C) The number of features in the dataset

D) The accuracy of the tree

Answer: **B) The impurity or "disagreement" in a set of examples** — Higher entropy means more mixed classes, lower entropy means more pure subsets.

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are $+/$-, return root with label $= +/$-

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are $+/$-, return root with label $= +/$-
- If attributes $=$ empty, return root with most common value of Target Attribute in Examples

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are $+/$-, return root with label $= +/$-
- If attributes $=$ empty, return root with most common value of Target Attribute in Examples
- Begin

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are $+/$-, return root with label $= +/$-
- If attributes $=$ empty, return root with most common value of Target Attribute in Examples
- Begin
  - A $\leftarrow$ attribute from Attributes which best classifies Examples

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are +/-, return root with label = +/-
- If attributes = empty, return root with most common value of Target Attribute in Examples
- Begin
  - A $\leftarrow$ attribute from Attributes which best classifies Examples
  - Root $\leftarrow$ A

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are +/-, return root with label = +/-
- If attributes = empty, return root with most common value of Target Attribute in Examples
- Begin
  - A $\leftarrow$ attribute from Attributes which best classifies Examples
  - Root $\leftarrow$ A
  - For each value (v) of A

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are +/-, return root with label = +/-
- If attributes = empty, return root with most common value of Target Attribute in Examples
- Begin
  - A $\leftarrow$ attribute from Attributes which best classifies Examples
  - Root $\leftarrow$ A
  - For each value (v) of A
    - Add new tree branch : A = v

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are +/-, return root with label = +/-
- If attributes = empty, return root with most common value of Target Attribute in Examples
- Begin
  - A $\leftarrow$ attribute from Attributes which best classifies Examples
  - Root $\leftarrow$ A
  - For each value (v) of A
    - Add new tree branch : A = v
    - Examples_v: subset of examples that A = v

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are +/-, return root with label = +/-
- If attributes = empty, return root with most common value of Target Attribute in Examples
- Begin
  - A $\leftarrow$ attribute from Attributes which best classifies Examples
  - Root $\leftarrow$ A
  - For each value (v) of A
    - Add new tree branch : A = v
    - Examples_v: subset of examples that A = v
    - If Examples_v is empty: add leaf with label = most common value of Target Attribute

ID3 (Examples, Target Attribute, Attributes)

- Create a root node for tree
- If all examples are +/-, return root with label = +/-
- If attributes = empty, return root with most common value of Target Attribute in Examples
- Begin
  - A $\leftarrow$ attribute from Attributes which best classifies Examples
  - Root $\leftarrow$ A
  - For each value (v) of A
    - Add new tree branch : A = v
    - Examples_v: subset of examples that A = v
    - If Examples_v is empty: add leaf with label = most common value of Target Attribute
    - Else: ID3 (Examples_v, Target attribute, Attributes - A)

Training Data

Day	Outlook	Temp	Humidity	Windy	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Entropy calculated

We have 14 examples in S: 5 No, 9 Yes

$$\begin{aligned}\text{Entropy}(S) &= -p_{\text{No}} \log_2 p_{\text{No}} - p_{\text{Yes}} \log_2 p_{\text{Yes}} \\ &= -\frac{5}{14} \log_2 \left(\frac{5}{14}\right) - \frac{9}{14} \log_2 \left(\frac{9}{14}\right) = 0.940\end{aligned}$$

Information Gain for Outlook

Outlook	Play
Sunny	No
Sunny	No
Overcast	Yes
Rain	Yes
Rain	Yes
Rain	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rain	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rain	No

Information Gain for Outlook

Information Gain for Outlook

Outlook	**Play**
Sunny	No
Sunny	No
Sunny	No
Sunny	Yes
Sunny	Yes

We have 2 Yes, 3 No Entropy $= -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$

Information Gain for Outlook

Outlook	Play
Sunny	No
Sunny	No
Sunny	No
Sunny	Yes
Sunny	Yes

We have 2 Yes, 3 No Entropy $= -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$

Outlook	Play
Overcast	Yes
Overcast	Yes
Overcast	Yes
Overcast	Yes

We have 4 Yes, 0 No Entropy $= 0$ (pure subset)

Information Gain for Outlook

Outlook	Play
Sunny	No
Sunny	No
Sunny	No
Sunny	Yes
Sunny	Yes

We have 2 Yes, 3 No Entropy $= -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$

Outlook	Play
Overcast	Yes
Overcast	Yes
Overcast	Yes
Overcast	Yes

We have 4 Yes, 0 No Entropy $= 0$ (pure subset)

Outlook	Play
Rain	Yes
Rain	Yes
Rain	No
Rain	Yes
Rain	No

We have 3 Yes, 2 No Entropy $= -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$

Information Gain

$$\text{Gain}(S, \text{Outlook}) = \text{Entropy}(S) - \sum_{v \in \{\text{Rain, Sunny, Overcast}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Information Gain

$$
\begin{aligned}
\mathrm{Gain}(S, \mathrm{Outlook}) = \mathrm{Entropy}(S) - & \\
& \sum_{v \in \{\mathrm{Rain,\ Sunny,\ Overcast}\}} \frac{|S_v|}{|S|} \mathrm{Entropy}(S_v) \\
& = 0.940 - \frac{5}{14} \times 0.971 - \frac{4}{14} \times 0 - \frac{5}{14} \times 0.971
\end{aligned}
$$

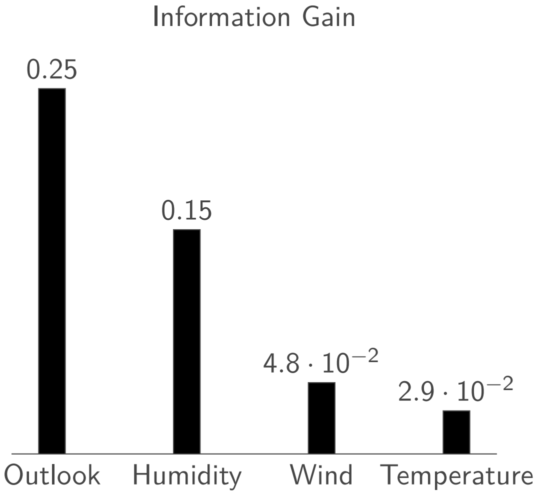
Information Gain

$$
\begin{aligned}
\text{Gain}(S, \text{Outlook}) &= \text{Entropy}(S) - \\
&\sum_{v \in \{\text{Rain, Sunny, Overcast}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \\
&= 0.940 - \frac{5}{14} \times 0.971 - \frac{4}{14} \times 0 - \frac{5}{14} \times 0.971 \\
&= 0.940 - 0.347 - 0 - 0.347
\end{aligned}
$$

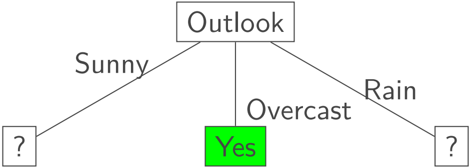
Information Gain

$$
\begin{aligned}
\text{Gain}(S, \text{Outlook}) &= \text{Entropy}(S) - \\
&\sum_{v \in \{\text{Rain, Sunny, Overcast}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \\
&= 0.940 - \frac{5}{14} \times 0.971 - \frac{4}{14} \times 0 - \frac{5}{14} \times 0.971 \\
&= 0.940 - 0.347 - 0 - 0.347 \\
&= 0.246
\end{aligned}
$$

Information Gain



Learnt Decision Tree



Calling ID3 on Outlook=Sunny

Day	Temp	Humidity	Windy	Play
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

- Gain($S_{\text{Outlook=Sunny}}$, Temp) = Entropy(2 Yes, 3 No) - (2/5)*Entropy(0 Yes, 2 No) -(2/5)*Entropy(1 Yes, 1 No) - (1/5)*Entropy(1 Yes, 0 No)

Calling ID3 on Outlook=Sunny

Day	Temp	Humidity	Windy	Play
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

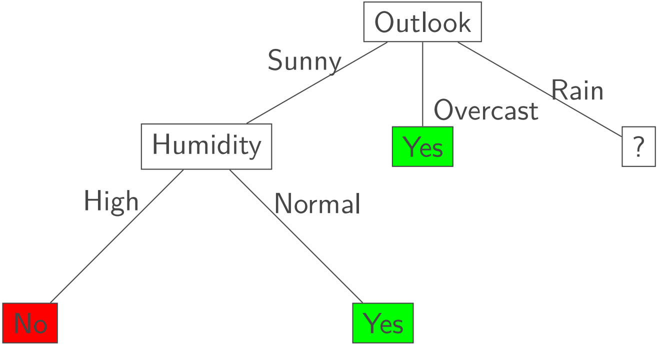
- Gain($S_{\text{Outlook=Sunny}}$, Temp) = Entropy(2 Yes, 3 No) - (2/5)*Entropy(0 Yes, 2 No) -(2/5)*Entropy(1 Yes, 1 No) - (1/5)*Entropy(1 Yes, 0 No)
- Gain($S_{\text{Outlook=Sunny}}$, Humidity) = Entropy(2 Yes, 3 No) - (2/5)*Entropy(2 Yes, 0 No) -(3/5)*Entropy(0 Yes, 3 No) $\Longrightarrow$ **maximum possible for the set**

Calling ID3 on Outlook=Sunny

Day	Temp	Humidity	Windy	Play
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

- Gain($S_{\text{Outlook=Sunny}}$, Temp) = Entropy(2 Yes, 3 No) - (2/5)*Entropy(0 Yes, 2 No) -(2/5)*Entropy(1 Yes, 1 No) - (1/5)*Entropy(1 Yes, 0 No)
- Gain($S_{\text{Outlook=Sunny}}$, Humidity) = Entropy(2 Yes, 3 No) - (2/5)*Entropy(2 Yes, 0 No) -(3/5)*Entropy(0 Yes, 3 No) $\Longrightarrow$ **maximum possible for the set**
- Gain($S_{\text{Outlook=Sunny}}$, Windy) = Entropy(2 Yes, 3 No) - (3/5)*Entropy(1 Yes, 2 No) -(2/5)*Entropy(1 Yes, 1 No)

Learnt Decision Tree

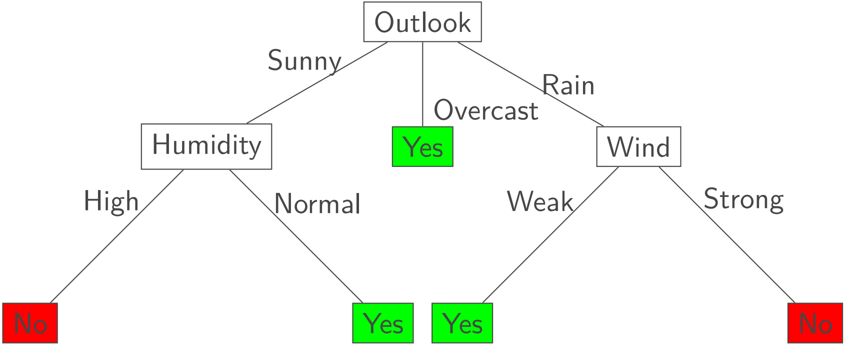


Calling ID3 on (Outlook=Rain)

Day	Temp	Humidity	Windy	Play
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

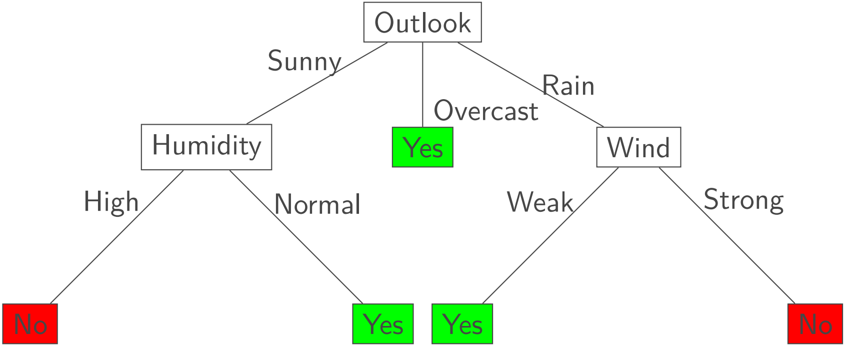
- The attribute Windy gives the highest information gain

Learnt Decision Tree



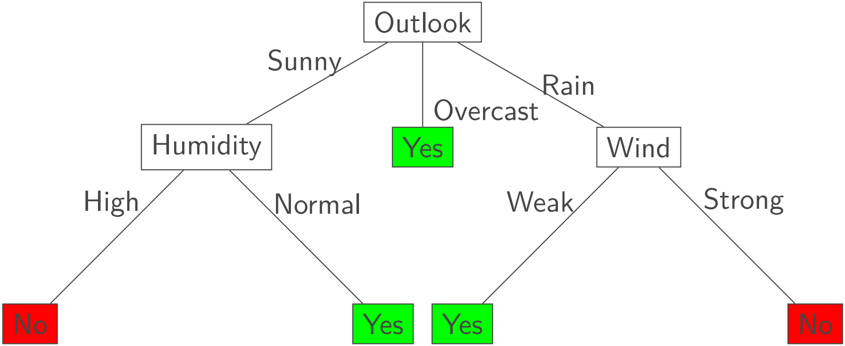
Prediction for Decision Tree

Just walk down the tree!



Prediction for Decision Tree

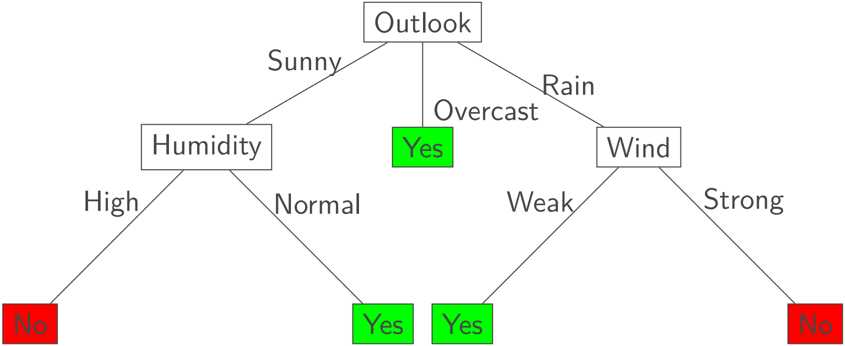
Just walk down the tree!



Prediction for <High Humidity, Strong Wind, Sunny Outlook, Hot Temp> is ?

Prediction for Decision Tree

Just walk down the tree!



Prediction for <High Humidity, Strong Wind, Sunny Outlook, Hot Temp> is ?

No

Limiting Tree Depth

Definition: Depth-Limited Trees

When depth limit is reached, assign the **most common class** in that path as the leaf node prediction.

Limiting Tree Depth

> **Definition: Depth-Limited Trees**
>
> When depth limit is reached, assign the **most common class** in that path as the leaf node prediction.

- **Depth-0 tree** (no decisions):

Limiting Tree Depth

Definition: Depth-Limited Trees

When depth limit is reached, assign the **most common class** in that path as the leaf node prediction.

- **Depth-0 tree** (no decisions):
  - Always predict the most common class

Limiting Tree Depth

Definition: Depth-Limited Trees

When depth limit is reached, assign the **most common class** in that path as the leaf node prediction.

- **Depth-0 tree** (no decisions):
  - Always predict the most common class
  - For our dataset: Always predict **Yes**

Limiting Tree Depth

Definition: Depth-Limited Trees

When depth limit is reached, assign the **most common class** in that path as the leaf node prediction.

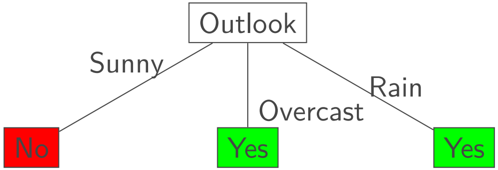
- **Depth-0 tree** (no decisions):
  - Always predict the most common class
  - For our dataset: Always predict **Yes**
- **Depth-1 tree** (single decision):

Limiting Tree Depth

Definition: Depth-Limited Trees

When depth limit is reached, assign the **most common class** in that path as the leaf node prediction.

- **Depth-0 tree** (no decisions):
  - Always predict the most common class
  - For our dataset: Always predict **Yes**
- **Depth-1 tree** (single decision):



Pop Quiz #3

Answer this!

In the tennis dataset, why did “Outlook” have the highest information gain?

A) It was the first feature in the dataset
B) When Outlook=Overcast, all examples have Play=Yes (pure subset)
C) It has the most possible values
D) It was chosen randomly

Pop Quiz #3

Answer this!

In the tennis dataset, why did "Outlook" have the highest information gain?

A) It was the first feature in the dataset

B) When Outlook=Overcast, all examples have Play=Yes (pure subset)

C) It has the most possible values

D) It was chosen randomly

Answer: **B) When Outlook=Overcast, all examples have Play=Yes** - This creates a pure subset with entropy=0, maximizing information gain.

Discrete Input, Real Output

Modified Dataset

Day	Outlook	Temp	Humidity	Wind	Minutes Played
D1	Sunny	Hot	High	Weak	20
D2	Sunny	Hot	High	Strong	24
D3	Overcast	Hot	High	Weak	40
D4	Rain	Mild	High	Weak	50
D5	Rain	Cool	Normal	Weak	60
D6	Rain	Cool	Normal	Strong	10
D7	Overcast	Cool	Normal	Strong	4
D8	Sunny	Mild	High	Weak	10
D9	Sunny	Cool	Normal	Weak	60
D10	Rain	Mild	Normal	Weak	40
D11	Sunny	Mild	High	Strong	45
D12	Overcast	Mild	High	Strong	40
D13	Overcast	Hot	Normal	Weak	35
D14	Rain	Mild	High	Strong	20

Regression Trees: From Classification to Regression

- Classification trees predict discrete classes (Yes/No, categories)

Regression Trees: From Classification to Regression

- Classification trees predict discrete classes (Yes/No, categories)
- Regression trees predict continuous numeric values

Regression Trees: From Classification to Regression

- Classification trees predict discrete classes (Yes/No, categories)
- Regression trees predict continuous numeric values
- **Key Question:** How do we measure impurity for continuous outputs?

Regression Trees: From Classification to Regression

- Classification trees predict discrete classes (Yes/No, categories)
- Regression trees predict continuous numeric values
- **Key Question:** How do we measure impurity for continuous outputs?
- For classification: Used entropy, information gain

Regression Trees: From Classification to Regression

- Classification trees predict discrete classes (Yes/No, categories)
- Regression trees predict continuous numeric values
- **Key Question:** How do we measure impurity for continuous outputs?
- For classification: Used entropy, information gain
- For regression: Use **Mean Squared Error (MSE)**

Regression Trees: From Classification to Regression

- Classification trees predict discrete classes (Yes/No, categories)
- Regression trees predict continuous numeric values
- **Key Question:** How do we measure impurity for continuous outputs?
- For classification: Used entropy, information gain
- For regression: Use **Mean Squared Error (MSE)**

Key Points

Why MSE for Regression?
MSE measures how far predicted values are from actual values.
Lower MSE = Better predictions = Less "impurity" in the data

Mean Squared Error (MSE): The Mathematics

Definition: Mean Squared Error

For a dataset S with n data points and target values $y_1, y_2, \ldots, y_n$:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the mean of target values

Mean Squared Error (MSE): The Mathematics

Definition: Mean Squared Error

For a dataset S with n data points and target values $y_1, y_2, \ldots, y_n$:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the mean of target values

- $(y_i - \bar{y})^2$: Squared difference between actual and mean

Mean Squared Error (MSE): The Mathematics

Definition: Mean Squared Error

For a dataset S with n data points and target values $y_1, y_2, \ldots, y_n$:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the mean of target values

- $(y_i - \bar{y})^2$: Squared difference between actual and mean
- Squaring ensures positive values and penalizes large errors

Mean Squared Error (MSE): The Mathematics

Definition: Mean Squared Error

For a dataset S with n data points and target values $y_1, y_2, \ldots, y_n$:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the mean of target values

- $(y_i - \bar{y})^2$: Squared difference between actual and mean
- Squaring ensures positive values and penalizes large errors
- MSE $= 0$ when all values are identical (perfect homogeneity)

Mean Squared Error (MSE): The Mathematics

Definition: Mean Squared Error

For a dataset S with n data points and target values $y_1, y_2, \ldots, y_n$:

$$\mathrm{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the mean of target values

- $(y_i - \bar{y})^2$: Squared difference between actual and mean
- Squaring ensures positive values and penalizes large errors
- MSE $= 0$ when all values are identical (perfect homogeneity)
- Higher MSE $=$ More variation $=$ Higher impurity

MSE Calculation: Step 1 - The Complete Dataset

Wind	Minutes Played
Weak	20
Strong	24
Weak	40
Weak	50
Weak	60
Strong	10
Strong	4
Weak	10
Weak	60
Weak	40
Strong	45
Strong	40
Weak	35
Strong	20

- **Tennis Dataset:** Predicting minutes played (continuous target)

MSE Calculation: Step 1 - The Complete Dataset

Wind	Minutes Played
Weak	20
Strong	24
Weak	40
Weak	50
Weak	60
Strong	10
Strong	4
Weak	10
Weak	60
Weak	40
Strong	45
Strong	40
Weak	35
Strong	20

- **Tennis Dataset:** Predicting minutes played (continuous target)
- **Goal:** Calculate MSE for the entire dataset S

MSE Calculation: Step 1 - The Complete Dataset

Wind	Minutes Played
Weak	20
Strong	24
Weak	40
Weak	50
Weak	60
Strong	10
Strong	4
Weak	10
Weak	60
Weak	40
Strong	45
Strong	40
Weak	35
Strong	20

- **Tennis Dataset:** Predicting minutes played (continuous target)
- **Goal:** Calculate MSE for the entire dataset S
- **Step 1:** Find the mean $\bar{y}$ of all target values

MSE Calculation: Step 2 - Computing the Mean

Example: Calculating Mean Minutes Played

All target values: 20, 24, 40, 50, 60, 10, 4, 10, 60, 40, 45, 40, 35, 20

MSE Calculation: Step 2 - Computing the Mean

Example: Calculating Mean Minutes Played

All target values: 20, 24, 40, 50, 60, 10, 4, 10, 60, 40, 45, 40, 35, 20

Step 1: Sum all values

$$\sum y_i = 20 + 24 + 40 + 50 + 60 + 10 + 4 + 10 + 60 + 40 + 45 + 40 + 35 + 20$$

MSE Calculation: Step 2 - Computing the Mean

Example: Calculating Mean Minutes Played

All target values: 20, 24, 40, 50, 60, 10, 4, 10, 60, 40, 45, 40, 35, 20

Step 1: Sum all values

$$\begin{aligned}\sum y_i &= 20 + 24 + 40 + 50 + 60 + 10 + 4 + 10 \\ &\quad + 60 + 40 + 45 + 40 + 35 + 20 \\ &= 458\end{aligned}$$

MSE Calculation: Step 2 - Computing the Mean

Example: Calculating Mean Minutes Played

All target values: 20, 24, 40, 50, 60, 10, 4, 10, 60, 40, 45, 40, 35, 20

Step 1: Sum all values

$$\begin{aligned} \sum y_i &= 20 + 24 + 40 + 50 + 60 + 10 + 4 + 10 \\ &\quad + 60 + 40 + 45 + 40 + 35 + 20 \\ &= 458 \end{aligned}$$

Step 2: Divide by number of data points ($n = 14$)

$$\bar{y} = \frac{458}{14} = 32.71 \text{ minutes}$$

MSE Calculation: Step 3 - Computing Squared Differences

Example: Calculating $(y_i - \bar{y})^2$ for Each Data Point

With $\bar{y} = 32.71$:

y_i	$y_i - \bar{y}$	$(y_i - \bar{y})^2$
20	$20 - 32.71 = -12.71$	$(-12.71)^2 = 161.54$
24	$24 - 32.71 = -8.71$	$(-8.71)^2 = 75.86$
40	$40 - 32.71 = 7.29$	$(7.29)^2 = 53.14$
50	$50 - 32.71 = 17.29$	$(17.29)^2 = 299.14$
60	$60 - 32.71 = 27.29$	$(27.29)^2 = 744.74$
10	$10 - 32.71 = -22.71$	$(-22.71)^2 = 515.74$
4	$4 - 32.71 = -28.71$	$(-28.71)^2 = 824.26$

MSE Calculation: Step 3 - Computing Squared Differences

Example: Calculating $(y_i - \bar{y})^2$ for Each Data Point

With $\bar{y} = 32.71$:

y_i	$y_i - \bar{y}$	$(y_i - \bar{y})^2$
20	$20 - 32.71 = -12.71$	$(-12.71)^2 = 161.54$
24	$24 - 32.71 = -8.71$	$(-8.71)^2 = 75.86$
40	$40 - 32.71 = 7.29$	$(7.29)^2 = 53.14$
50	$50 - 32.71 = 17.29$	$(17.29)^2 = 299.14$
60	$60 - 32.71 = 27.29$	$(27.29)^2 = 744.74$
10	$10 - 32.71 = -22.71$	$(-22.71)^2 = 515.74$
4	$4 - 32.71 = -28.71$	$(-28.71)^2 = 824.26$

Continue this for all 14 data points...

MSE Calculation: Step 4 - Complete Squared Differences

Example: All Squared Differences

y_i	$y_i - \bar{y}$	$(y_i - \bar{y})^2$
20	-12.71	161.54
24	-8.71	75.86
40	7.29	53.14
50	17.29	299.14
60	27.29	744.74
10	-22.71	515.74
4	-28.71	824.26
10	-22.71	515.74
60	27.29	744.74
40	7.29	53.14
45	12.29	151.04
40	7.29	53.14
35	2.29	5.24
20	-12.71	161.54
Sum		**4358.86**

MSE Calculation: Step 5 - Final MSE Computation

Example: Computing MSE for Complete Dataset

Formula:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

MSE Calculation: Step 5 - Final MSE Computation

Example: Computing MSE for Complete Dataset

Formula:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Substituting our values:

$$\text{MSE}(S) = \frac{1}{14} \times 4358.86 = 311.35$$

MSE Calculation: Step 5 - Final MSE Computation

Example: Computing MSE for Complete Dataset

Formula:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Substituting our values:

$$\text{MSE}(S) = \frac{1}{14} \times 4358.86 = 311.35$$

Interpretation:

- MSE $= 311.35$ square-minutes

MSE Calculation: Step 5 - Final MSE Computation

Example: Computing MSE for Complete Dataset

Formula:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Substituting our values:

$$\text{MSE}(S) = \frac{1}{14} \times 4358.86 = 311.35$$

Interpretation:

- MSE $= 311.35$ square-minutes
- This measures the "impurity" or variation in our dataset

MSE Calculation: Step 5 - Final MSE Computation

Example: Computing MSE for Complete Dataset

Formula:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Substituting our values:

$$\text{MSE}(S) = \frac{1}{14} \times 4358.86 = 311.35$$

Interpretation:

- MSE $=$ 311.35 square-minutes
- This measures the "impurity" or variation in our dataset
- Higher MSE $=$ More variation in target values

MSE Calculation: Step 5 - Final MSE Computation

Example: Computing MSE for Complete Dataset

Formula:

$$\text{MSE}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Substituting our values:

$$\text{MSE}(S) = \frac{1}{14} \times 4358.86 = 311.35$$

Interpretation:

- MSE = 311.35 square-minutes
- This measures the "impurity" or variation in our dataset
- Higher MSE = More variation in target values
- When we split the data, we want to reduce this MSE

MSE Reduction: The Splitting Criterion

Definition: MSE Reduction Formula

For a split on attribute A with values $v_1, v_2, \ldots, v_k$:

$$\text{MSE Reduction} = \text{MSE}(S) - \sum_{j=1}^{k} \frac{|S_{v_j}|}{|S|} \times \text{MSE}(S_{v_j})$$

where:

- S is the original dataset

MSE Reduction: The Splitting Criterion

Definition: MSE Reduction Formula

For a split on attribute A with values $v_1, v_2, \ldots, v_k$:

$$\text{MSE Reduction} = \text{MSE}(S) - \sum_{j=1}^{k} \frac{|S_{v_j}|}{|S|} \times \text{MSE}(S_{v_j})$$

where:

- S is the original dataset
- S_{v_j} is the subset with attribute value v_j

MSE Reduction: The Splitting Criterion

Definition: MSE Reduction Formula

For a split on attribute A with values $v_1, v_2, \ldots, v_k$:

$$\text{MSE Reduction} = \text{MSE}(S) - \sum_{j=1}^{k} \frac{|S_{v_j}|}{|S|} \times \text{MSE}(S_{v_j})$$

where:

- S is the original dataset
- S_{v_j} is the subset with attribute value v_j
- $|S_{v_j}|$ is the size of subset S_{v_j}

MSE Reduction: The Splitting Criterion

Definition: MSE Reduction Formula

For a split on attribute A with values $v_1, v_2, \ldots, v_k$:

$$\text{MSE Reduction} = \text{MSE}(S) - \sum_{j=1}^{k} \frac{|S_{v_j}|}{|S|} \times \text{MSE}(S_{v_j})$$

where:

- S is the original dataset
- S_{v_j} is the subset with attribute value v_j
- $|S_{v_j}|$ is the size of subset S_{v_j}
- $|S|$ is the size of original dataset

MSE Reduction: The Splitting Criterion

Definition: MSE Reduction Formula

For a split on attribute A with values $v_1, v_2, \ldots, v_k$:

$$\text{MSE Reduction} = \text{MSE}(S) - \sum_{j=1}^{k} \frac{|S_{v_j}|}{|S|} \times \text{MSE}(S_{v_j})$$

where:

- S is the original dataset
- S_{v_j} is the subset with attribute value v_j
- $|S_{v_j}|$ is the size of subset S_{v_j}
- $|S|$ is the size of original dataset

Key Points

Key Insight: MSE Reduction > 0 means the split improves our model!
Choose the split with highest MSE Reduction

Splitting on Wind: Step 1 - Partition the Data

Splitting on Wind: Step 1 - Partition the Data

Example: Wind = Weak (8 points)

Wind	Minutes
Weak	20
Weak	40
Weak	50
Weak	60
Weak	10
Weak	60
Weak	40
Weak	35

Splitting on Wind: Step 1 - Partition the Data

Example: Wind = Weak (8 points)

Wind	Minutes
Weak	20
Weak	40
Weak	50
Weak	60
Weak	10
Weak	60
Weak	40
Weak	35

Example: Wind = Strong (6 points)

Wind	Minutes
Strong	24
Strong	10
Strong	4
Strong	45
Strong	40
Strong	20

Splitting on Wind: Step 1 - Partition the Data

Example: Wind = Weak (8 points)

Wind	Minutes
Weak	20
Weak	40
Weak	50
Weak	60
Weak	10
Weak	60
Weak	40
Weak	35

Example: Wind = Strong (6 points)

Wind	Minutes
Strong	24
Strong	10
Strong	4
Strong	45
Strong	40
Strong	20

- **Original dataset:** 14 points, MSE = 311.35

Splitting on Wind: Step 1 - Partition the Data

Example: Wind = Weak (8 points)

Wind	Minutes
Weak	20
Weak	40
Weak	50
Weak	60
Weak	10
Weak	60
Weak	40
Weak	35

Example: Wind = Strong (6 points)

Wind	Minutes
Strong	24
Strong	10
Strong	4
Strong	45
Strong	40
Strong	20

- **Original dataset:** 14 points, MSE = 311.35
- **After split:** 8 points (Weak) + 6 points (Strong)

Splitting on Wind: Step 1 - Partition the Data

Example: Wind = Weak (8 points)

Wind	Minutes
Weak	20
Weak	40
Weak	50
Weak	60
Weak	10
Weak	60
Weak	40
Weak	35

Example: Wind = Strong (6 points)

Wind	Minutes
Strong	24
Strong	10
Strong	4
Strong	45
Strong	40
Strong	20

- **Original dataset:** 14 points, MSE = 311.35
- **After split:** 8 points (Weak) + 6 points (Strong)
- **Next:** Calculate MSE for each subset

Splitting on Wind: Step 2 - MSE for Wind=Weak

Example: Calculating $\mathsf{MSE}(S_{\textbf{Wind=Weak}})$

Data points: 20, 40, 50, 60, 10, 60, 40, 35

Splitting on Wind: Step 2 - MSE for Wind=Weak

Example: Calculating $\mathrm{MSE}(S_{\textbf{Wind=Weak}})$

Data points: 20, 40, 50, 60, 10, 60, 40, 35
Step 1: Calculate mean

$$\begin{aligned}\bar{y}_{\text{weak}} &= \frac{20+40+50+60+10+60+40+35}{8} \\ &= \frac{315}{8} = 39.375\end{aligned}$$

Splitting on Wind: Step 2 - MSE for Wind=Weak

Example: Calculating $\text{MSE}(S_{\textbf{Wind=Weak}})$

Data points: 20, 40, 50, 60, 10, 60, 40, 35

Step 1: Calculate mean

$$\bar{y}_{\text{weak}} = \frac{20 + 40 + 50 + 60 + 10 + 60 + 40 + 35}{8}$$
$$= \frac{315}{8} = 39.375$$

Step 2: Calculate squared differences

y_i	$y_i - 39.375$	$(y_i - 39.375)^2$
20	-19.375	375.39
40	0.625	0.39
50	10.625	112.89
60	20.625	425.39
10	-29.375	862.89
60	20.625	425.39
40	0.625	0.39
35	-4.375	19.14
Sum		**2221.87**

Splitting on Wind: Step 3 - Complete MSE for Wind=Weak

Example: Final MSE Calculation for Wind=Weak

$$\text{MSE}(S_{\text{Wind}=\text{Weak}}) = \frac{1}{8} \times 2221.87 = 277.73$$

Splitting on Wind: Step 3 - Complete MSE for Wind=Weak

Example: Final MSE Calculation for Wind=Weak

$$\text{MSE}(S_{\text{Wind=Weak}}) = \frac{1}{8} \times 2221.87 = 277.73$$

Example: Verification Check

- Original MSE for all data: 311.35

Splitting on Wind: Step 3 - Complete MSE for Wind=Weak

Example: Final MSE Calculation for Wind=Weak

$$\text{MSE}(S_{\text{Wind=Weak}}) = \frac{1}{8} \times 2221.87 = 277.73$$

Example: Verification Check

- Original MSE for all data: 311.35
- MSE for Wind=Weak subset: 277.73

Splitting on Wind: Step 3 - Complete MSE for Wind=Weak

Example: Final MSE Calculation for Wind=Weak

$$\text{MSE}(S_{\text{Wind=Weak}}) = \frac{1}{8} \times 2221.87 = 277.73$$

Example: Verification Check

- Original MSE for all data: 311.35
- MSE for Wind=Weak subset: 277.73
- **Good sign:** MSE decreased (less variation within this group)

Splitting on Wind: Step 3 - Complete MSE for Wind=Weak

Example: Final MSE Calculation for Wind=Weak

$$\text{MSE}(S_{\text{Wind=Weak}}) = \frac{1}{8} \times 2221.87 = 277.73$$

Example: Verification Check

- Original MSE for all data: 311.35
- MSE for Wind=Weak subset: 277.73
- **Good sign:** MSE decreased (less variation within this group)
- This subset is more "homogeneous" than the full dataset

Splitting on Wind: Step 4 - MSE for Wind=Strong

Example: Calculating $\mathrm{MSE}(S_{\textbf{Wind=Strong}})$

Data points: 24, 10, 4, 45, 40, 20

Splitting on Wind: Step 4 - MSE for Wind=Strong

Example: Calculating $\mathrm{MSE}(S_{\textbf{Wind=Strong}})$

Data points: 24, 10, 4, 45, 40, 20
Step 1: Calculate mean

$$\bar{y}_{\text{strong}} = \frac{24 + 10 + 4 + 45 + 40 + 20}{6} = \frac{143}{6} = 23.83$$

Splitting on Wind: Step 4 - MSE for Wind=Strong

Example: Calculating $\mathrm{MSE}(S_{\textbf{Wind=Strong}})$

Data points: 24, 10, 4, 45, 40, 20

Step 1: Calculate mean

$$\bar{y}_{\text{strong}} = \frac{24 + 10 + 4 + 45 + 40 + 20}{6} = \frac{143}{6} = 23.83$$

Step 2: Calculate squared differences

y_i	$y_i - 23.83$	$(y_i - 23.83)^2$
24	0.17	0.03
10	-13.83	191.27
4	-19.83	393.23
45	21.17	448.17
40	16.17	261.47
20	-3.83	14.67
Sum		**1308.84**

Splitting on Wind: Step 4 - MSE for Wind=Strong

Example: Calculating MSE($S_{\text{Wind=Strong}}$)

Data points: 24, 10, 4, 45, 40, 20

Step 1: Calculate mean

$$\bar{y}_{\text{strong}} = \frac{24 + 10 + 4 + 45 + 40 + 20}{6} = \frac{143}{6} = 23.83$$

Step 2: Calculate squared differences

y_i	$y_i - 23.83$	$(y_i - 23.83)^2$
24	0.17	0.03
10	-13.83	191.27
4	-19.83	393.23
45	21.17	448.17
40	16.17	261.47
20	-3.83	14.67
Sum		**1308.84**

$$\text{MSE}(S_{\text{Wind=Strong}}) = \frac{1}{6} \times 1308.84 = 218.14$$

Splitting on Wind: Step 5 - Computing MSE Reduction

Example: Final MSE Reduction Calculation

We have:

- $\text{MSE}(S) = 311.35$ (original dataset)

Splitting on Wind: Step 5 - Computing MSE Reduction

Example: Final MSE Reduction Calculation

We have:

- $\text{MSE}(S) = 311.35$ (original dataset)
- $\text{MSE}(S_{\text{Wind=Weak}}) = 277.73$ (8 points)

Splitting on Wind: Step 5 - Computing MSE Reduction

Example: Final MSE Reduction Calculation

We have:

- $\text{MSE}(S) = 311.35$ (original dataset)
- $\text{MSE}(S_{\text{Wind=Weak}}) = 277.73$ (8 points)
- $\text{MSE}(S_{\text{Wind=Strong}}) = 218.14$ (6 points)

Splitting on Wind: Step 5 - Computing MSE Reduction

Example: Final MSE Reduction Calculation

We have:

- $\text{MSE}(S) = 311.35$ (original dataset)
- $\text{MSE}(S_{\text{Wind=Weak}}) = 277.73$ (8 points)
- $\text{MSE}(S_{\text{Wind=Strong}}) = 218.14$ (6 points)

Weighted Average MSE:

$$\begin{aligned}\text{Weighted MSE} &= \frac{8}{14} \times 277.73 + \frac{6}{14} \times 218.14 \\ &= 0.571 \times 277.73 + 0.429 \times 218.14 \\ &= 158.60 + 93.58 = 252.18\end{aligned}$$

Splitting on Wind: Step 5 - Computing MSE Reduction

Example: Final MSE Reduction Calculation

We have:

- $\text{MSE}(S) = 311.35$ (original dataset)
- $\text{MSE}(S_{\text{Wind=Weak}}) = 277.73$ (8 points)
- $\text{MSE}(S_{\text{Wind=Strong}}) = 218.14$ (6 points)

Weighted Average MSE:

$$\begin{aligned}
\text{Weighted MSE} &= \frac{8}{14} \times 277.73 + \frac{6}{14} \times 218.14 \\
&= 0.571 \times 277.73 + 0.429 \times 218.14 \\
&= 158.60 + 93.58 = 252.18
\end{aligned}$$

MSE Reduction:

$$\text{MSE Reduction} = 311.35 - 252.18 = \mathbf{59.17}$$

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!
- **Magnitude:** We reduced prediction error by 59.17 square-minutes

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!
- **Magnitude:** We reduced prediction error by 59.17 square-minutes
- **Percentage:** $(59.17/311.35) \times 100\% = 19\%$ improvement

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!
- **Magnitude:** We reduced prediction error by 59.17 square-minutes
- **Percentage:** $(59.17/311.35) \times 100\% = 19\%$ improvement
- **Intuition:** Wind attribute helps separate high/low playing minutes

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!
- **Magnitude:** We reduced prediction error by 59.17 square-minutes
- **Percentage:** $(59.17/311.35) \times 100\% = 19\%$ improvement
- **Intuition:** Wind attribute helps separate high/low playing minutes

Example: Decision Tree Building Process

- **Step 1:** Calculate MSE reduction for all possible splits

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!
- **Magnitude:** We reduced prediction error by 59.17 square-minutes
- **Percentage:** $(59.17/311.35) \times 100\% = 19\%$ improvement
- **Intuition:** Wind attribute helps separate high/low playing minutes

Example: Decision Tree Building Process

- **Step 1:** Calculate MSE reduction for all possible splits
- **Step 2:** Choose the split with *highest* MSE reduction

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!
- **Magnitude:** We reduced prediction error by 59.17 square-minutes
- **Percentage:** $(59.17/311.35) \times 100\% = 19\%$ improvement
- **Intuition:** Wind attribute helps separate high/low playing minutes

Example: Decision Tree Building Process

- **Step 1:** Calculate MSE reduction for all possible splits
- **Step 2:** Choose the split with *highest* MSE reduction
- **Step 3:** Recursively apply to child nodes

MSE Reduction: Interpretation and Decision Making

Key Points

What Does MSE Reduction = 59.17 Mean?

- **Positive value:** The split improves our model!
- **Magnitude:** We reduced prediction error by 59.17 square-minutes
- **Percentage:** $(59.17/311.35) \times 100\% = 19\%$ improvement
- **Intuition:** Wind attribute helps separate high/low playing minutes

Example: Decision Tree Building Process

- **Step 1:** Calculate MSE reduction for all possible splits
- **Step 2:** Choose the split with *highest* MSE reduction
- **Step 3:** Recursively apply to child nodes
- **Stop when:** MSE reduction becomes too small or max depth reached

MSE Reduction: Interpretation and Decision Making

Example: Decision Tree Building Process

- **Step 1:** Calculate MSE reduction for all possible splits
- **Step 2:** Choose the split with *highest* MSE reduction
- **Step 3:** Recursively apply to child nodes
- **Stop when:** MSE reduction becomes too small or max depth reached

MSE Reduction: Interpretation and Decision Making

Example: Decision Tree Building Process

- **Step 1:** Calculate MSE reduction for all possible splits
- **Step 2:** Choose the split with *highest* MSE reduction
- **Step 3:** Recursively apply to child nodes
- **Stop when:** MSE reduction becomes too small or max depth reached

Important: Key Difference from Classification

Classification: Use Information Gain (maximize information)
Regression: Use MSE Reduction (minimize prediction error)

Pop Quiz #5

Answer this!

For regression trees, what criterion do we use instead of Information Gain?

A) Information Gain
B) Gini Impurity
C) Mean Squared Error (MSE) Reduction
D) Accuracy

Pop Quiz #5

Answer this!

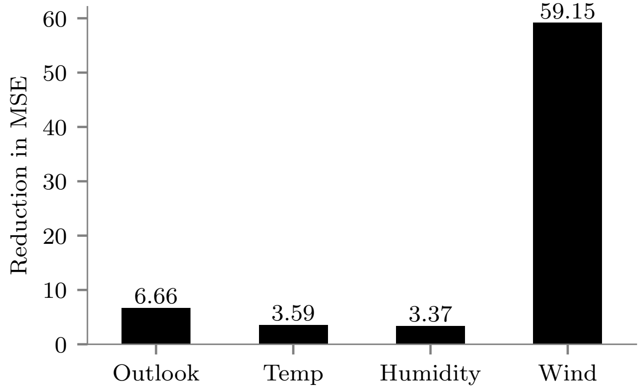
For regression trees, what criterion do we use instead of Information Gain?

A) Information Gain
B) Gini Impurity
C) Mean Squared Error (MSE) Reduction
D) Accuracy

Answer: **C) Mean Squared Error (MSE) Reduction**
- For regression, we minimize MSE instead of maximizing information gain.

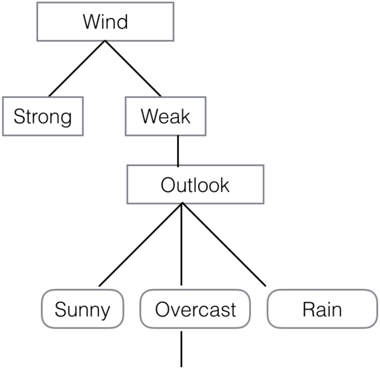
MSE Reduction for Regression Trees

Notebook: decision-tree-real-output.html



Learnt Tree

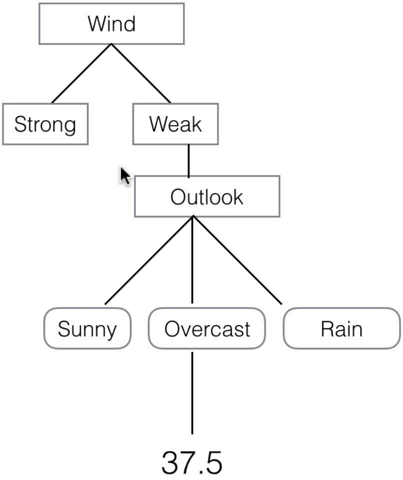
Assume a tree like this is learnt …



	Day	Outlook	Temp	Humidity	Wind	Minutes Played
2	D3	Overcast	Hot	High	Weak	40
12	D13	Overcast	Hot	Normal	Weak	35

Learnt Tree

Method 1
Mins
Played=(40+35)
/2



Real Input, Discrete Output

Moving to Our Third Case

Key Points

Our Journey Through Decision Tree Types:

- **Discrete Input, Discrete Output**: Simple categorical splits

What's different now?

Moving to Our Third Case

Key Points

Our Journey Through Decision Tree Types:

- **Discrete Input, Discrete Output**: Simple categorical splits
- **Real Input, Real Output**: Continuous features, regression trees

What's different now?

Moving to Our Third Case

Key Points

Our Journey Through Decision Tree Types:

- **Discrete Input, Discrete Output**: Simple categorical splits
- **Real Input, Real Output**: Continuous features, regression trees
- **Real Input, Discrete Output**: Continuous features, classification

What's different now?

Moving to Our Third Case

Key Points

Our Journey Through Decision Tree Types:

- **Discrete Input, Discrete Output**: Simple categorical splits
- **Real Input, Real Output**: Continuous features, regression trees
- **Real Input, Discrete Output**: Continuous features, classification

What's different now?

- Input: Continuous/real-valued features (like temperature, age, income)

Moving to Our Third Case

Key Points

Our Journey Through Decision Tree Types:

- **Discrete Input, Discrete Output**: Simple categorical splits
- **Real Input, Real Output**: Continuous features, regression trees
- **Real Input, Discrete Output**: Continuous features, classification

What's different now?

- Input: Continuous/real-valued features (like temperature, age, income)
- Output: Discrete classes (Yes/No, Low/Medium/High, etc.)

Moving to Our Third Case

Key Points

Our Journey Through Decision Tree Types:

- **Discrete Input, Discrete Output**: Simple categorical splits
- **Real Input, Real Output**: Continuous features, regression trees
- **Real Input, Discrete Output**: Continuous features, classification

What's different now?

- Input: Continuous/real-valued features (like temperature, age, income)
- Output: Discrete classes (Yes/No, Low/Medium/High, etc.)
- Challenge: Where exactly should we split the continuous feature?

The Key Challenge: Infinite Split Points

Important: The Problem

With continuous features, we have potentially infinite split points!

- Temperature could be split at 45°C, 45.1°C, 45.01°C, ...

The Intuitive Solution:

The Key Challenge: Infinite Split Points

Important: The Problem

With continuous features, we have potentially infinite split points!

- Temperature could be split at 45°C, 45.1°C, 45.01°C, ...
- We need a systematic approach to find the **best** split points

The Intuitive Solution:

The Key Challenge: Infinite Split Points

Important: The Problem

With continuous features, we have potentially infinite split points!

- Temperature could be split at 45°C, 45.1°C, 45.01°C, ...
- We need a systematic approach to find the **best** split points

The Intuitive Solution:

1. Look for "natural boundaries" between different classes

The Key Challenge: Infinite Split Points

Important: The Problem

With continuous features, we have potentially infinite split points!

- Temperature could be split at 45°C, 45.1°C, 45.01°C, ...
- We need a systematic approach to find the **best** split points

The Intuitive Solution:

1. Look for "natural boundaries" between different classes
2. Focus on points where class labels actually change

The Key Challenge: Infinite Split Points

Important: The Problem

With continuous features, we have potentially infinite split points!

- Temperature could be split at 45°C, 45.1°C, 45.01°C, ...
- We need a systematic approach to find the **best** split points

The Intuitive Solution:

1. Look for "natural boundaries" between different classes
2. Focus on points where class labels actually change
3. Test splits that maximize information gain

The Tennis Example - Setting the Stage

Scenario: Should we play tennis based on temperature?

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

Question: How do we find the best split point in this continuous temperature data?

The Sorting Intuition - Why Start Here?

Example: Why Sort the Data First?

- Sorting reveals the natural **class boundaries** in the data

Sorted Data: (already sorted in our example)

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

The Sorting Intuition - Why Start Here?

Example: Why Sort the Data First?

- Sorting reveals the natural **class boundaries** in the data
- We can see where labels change: No $\rightarrow$ Yes $\rightarrow$ No

Sorted Data: (already sorted in our example)

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

The Sorting Intuition - Why Start Here?

Example: Why Sort the Data First?

- Sorting reveals the natural **class boundaries** in the data
- We can see where labels change: No $\rightarrow$ Yes $\rightarrow$ No
- Only need to consider splits between different class labels

Sorted Data: (already sorted in our example)

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

The Sorting Intuition - Why Start Here?

Example: Why Sort the Data First?

- Sorting reveals the natural **class boundaries** in the data
- We can see where labels change: No $\rightarrow$ Yes $\rightarrow$ No
- Only need to consider splits between different class labels
- Eliminates millions of irrelevant split points!

Sorted Data: (already sorted in our example)

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

Finding Smart Split Points

Definition: The Midpoint Strategy

Only consider splits at **midpoints** between consecutive different classes:

- Between 48(No) and 60(Yes): split at $(48 + 60)/2 = 54$

All candidate splits: 44, 54, 66, 76, 85

Key Points

Why These 5 Splits?

Finding Smart Split Points

Definition: The Midpoint Strategy

Only consider splits at **midpoints** between consecutive different classes:

- Between 48(No) and 60(Yes): split at $(48 + 60)/2 = 54$
- Between 80(Yes) and 90(No): split at $(80 + 90)/2 = 85$

All candidate splits: 44, 54, 66, 76, 85

Key Points

Why These 5 Splits?

Finding Smart Split Points

Definition: The Midpoint Strategy

Only consider splits at **midpoints** between consecutive different classes:

- Between 48(No) and 60(Yes): split at $(48 + 60)/2 = 54$
- Between 80(Yes) and 90(No): split at $(80 + 90)/2 = 85$

All candidate splits: 44, 54, 66, 76, 85

Key Points

Why These 5 Splits?

- 44: Separates D1 from rest

Finding Smart Split Points

Definition: The Midpoint Strategy

Only consider splits at **midpoints** between consecutive different classes:

- Between 48(No) and 60(Yes): split at $(48 + 60)/2 = 54$
- Between 80(Yes) and 90(No): split at $(80 + 90)/2 = 85$

All candidate splits: 44, 54, 66, 76, 85

Key Points

Why These 5 Splits?

- 44: Separates D1 from rest
- 54: Separates No's from Yes's

Finding Smart Split Points

Definition: The Midpoint Strategy

Only consider splits at **midpoints** between consecutive different classes:

- Between 48(No) and 60(Yes): split at $(48 + 60)/2 = 54$
- Between 80(Yes) and 90(No): split at $(80 + 90)/2 = 85$

All candidate splits: 44, 54, 66, 76, 85

Key Points

Why These 5 Splits?

- 44: Separates D1 from rest
- 54: Separates No's from Yes's
- 66, 76: Split within the Yes region

Finding Smart Split Points

Definition: The Midpoint Strategy

Only consider splits at **midpoints** between consecutive different classes:

- Between 48(No) and 60(Yes): split at $(48 + 60)/2 = 54$
- Between 80(Yes) and 90(No): split at $(80 + 90)/2 = 85$

All candidate splits: 44, 54, 66, 76, 85

Key Points

Why These 5 Splits?

- 44: Separates D1 from rest
- 54: Separates No's from Yes's
- 66, 76: Split within the Yes region
- 85: Separates last Yes from final No

Evaluating Split at Temperature ≤ 44

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

Example: Split Analysis

Left side (Temp $\leq$ 44): 1 example, all "No" $\rightarrow$ Perfect purity!
Right side (Temp $>$ 44): 5 examples, 3 "Yes", 2 "No" $\rightarrow$ Mixed
Entropy(Left) $= 0$, Entropy(Right) $= 0.971$
Weighted Entropy $= \frac{1}{6} \times 0 + \frac{5}{6} \times 0.971 = 0.808$

Evaluating Split at Temperature ≤ 54

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

Example: Split Analysis

Left side (Temp $\leq$ 54): 2 examples, all "No" $\rightarrow$ Perfect purity!
Right side (Temp $>$ 54): 4 examples, 3 "Yes", 1 "No" $\rightarrow$ Better!
Entropy(Left) $= 0$, Entropy(Right) $= 0.811$
Weighted Entropy $= \frac{2}{6} \times 0 + \frac{4}{6} \times 0.811 = 0.541$

Comparing All Candidate Splits

Split Point	Left Side	Right Side	Weighted Entropy	Info Gain
44	1 No	3 Yes, 2 No	0.808	0.142
54	**2 No**	**3 Yes, 1 No**	**0.541**	**0.409**
66	2 No, 1 Yes	2 Yes, 1 No	0.918	0.032
76	2 No, 2 Yes	1 Yes, 1 No	1.000	-0.050
85	2 No, 3 Yes	1 No	0.650	0.300

Key Points

Winner: Split at 54!

- Lowest weighted entropy (0.541)

Comparing All Candidate Splits

Split Point	Left Side	Right Side	Weighted Entropy	Info Gain
44	1 No	3 Yes, 2 No	0.808	0.142
54	**2 No**	**3 Yes, 1 No**	**0.541**	**0.409**
66	2 No, 1 Yes	2 Yes, 1 No	0.918	0.032
76	2 No, 2 Yes	1 Yes, 1 No	1.000	-0.050
85	2 No, 3 Yes	1 No	0.650	0.300

Key Points

Winner: Split at 54!

- Lowest weighted entropy (0.541)
- Highest information gain (0.409)

Comparing All Candidate Splits

Split Point	Left Side	Right Side	Weighted Entropy	Info Gain
44	1 No	3 Yes, 2 No	0.808	0.142
54	**2 No**	**3 Yes, 1 No**	**0.541**	**0.409**
66	2 No, 1 Yes	2 Yes, 1 No	0.918	0.032
76	2 No, 2 Yes	1 Yes, 1 No	1.000	-0.050
85	2 No, 3 Yes	1 No	0.650	0.300

Key Points

Winner: Split at 54!

- Lowest weighted entropy (0.541)
- Highest information gain (0.409)
- Creates the best class separation

The Algorithm Summary

Definition: Decision Tree Algorithm for Continuous Features

1. **Sort** data by feature values

The Algorithm Summary

Definition: Decision Tree Algorithm for Continuous Features

1. **Sort** data by feature values
2. **Identify** candidate split points (midpoints between different classes)

The Algorithm Summary

Definition: Decision Tree Algorithm for Continuous Features

1. **Sort** data by feature values
2. **Identify** candidate split points (midpoints between different classes)
3. **Evaluate** each split using information gain:

The Algorithm Summary

Definition: Decision Tree Algorithm for Continuous Features

1. **Sort** data by feature values
2. **Identify** candidate split points (midpoints between different classes)
3. **Evaluate** each split using information gain:
    - Calculate weighted entropy for the split

The Algorithm Summary

Definition: Decision Tree Algorithm for Continuous Features

1. **Sort** data by feature values
2. **Identify** candidate split points (midpoints between different classes)
3. **Evaluate** each split using information gain:
   - Calculate weighted entropy for the split
   - Information Gain = Original Entropy - Weighted Entropy

The Algorithm Summary

Definition: Decision Tree Algorithm for Continuous Features

1. **Sort** data by feature values
2. **Identify** candidate split points (midpoints between different classes)
3. **Evaluate** each split using information gain:
   - Calculate weighted entropy for the split
   - Information Gain = Original Entropy - Weighted Entropy
4. **Choose** split with highest information gain

The Algorithm Summary

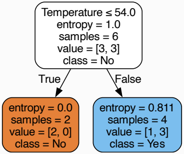
Definition: Decision Tree Algorithm for Continuous Features

1. **Sort** data by feature values
2. **Identify** candidate split points (midpoints between different classes)
3. **Evaluate** each split using information gain:
    - Calculate weighted entropy for the split
    - Information Gain = Original Entropy - Weighted Entropy
4. **Choose** split with highest information gain
5. **Recurse** on left and right subsets

Visual Example: The Resulting Decision Tree

Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

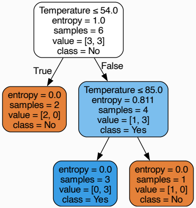
Notebook: decision-tree-real-input-discrete-output.html



Finding splits

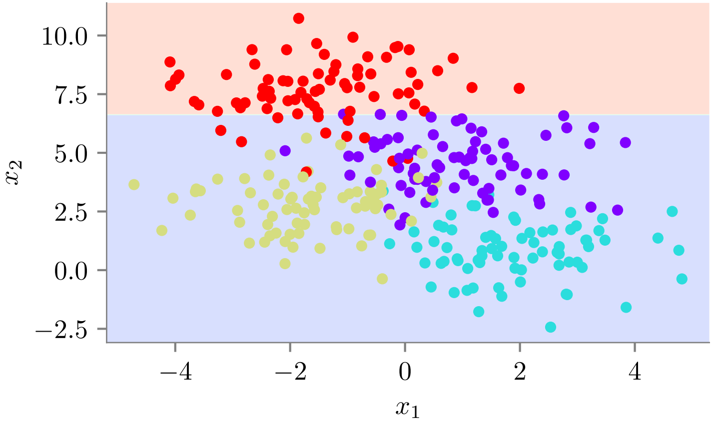
Day	Temperature	PlayTennis
D1	40	No
D2	48	No
D3	60	Yes
D4	72	Yes
D5	80	Yes
D6	90	No

Notebook: decision-tree-real-input-discrete-output.html



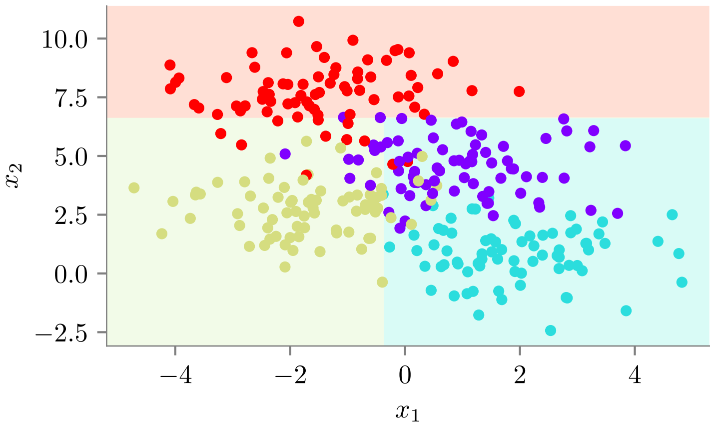
Example (DT of depth 1)

Notebook: decision-tree-real-input-discrete-output.html



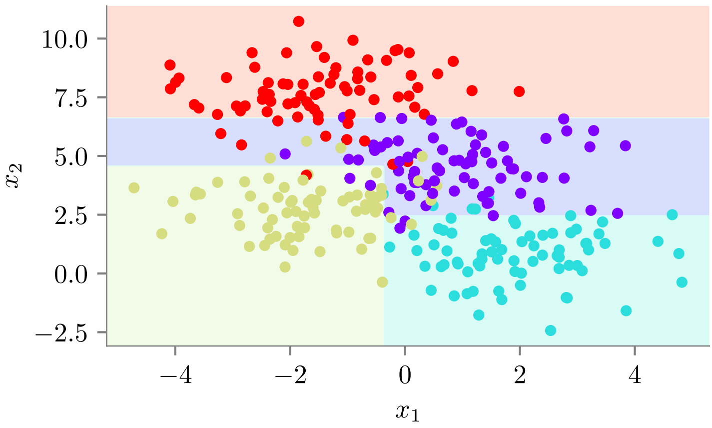
Example (DT of depth 2)

Notebook: decision-tree-real-input-discrete-output.html



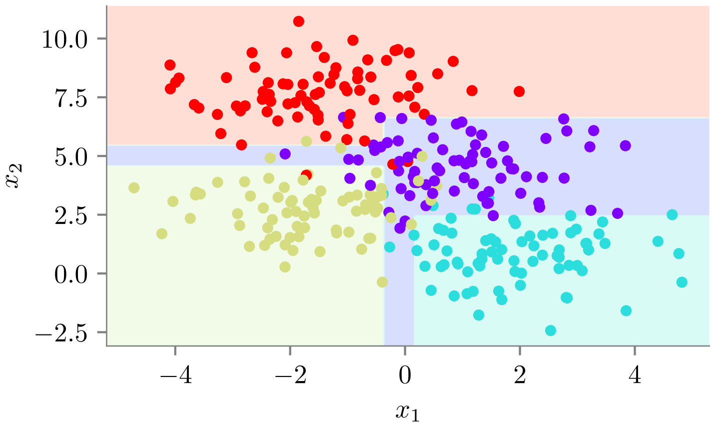
Example (DT of depth 3)

Notebook: decision-tree-real-input-discrete-output.html



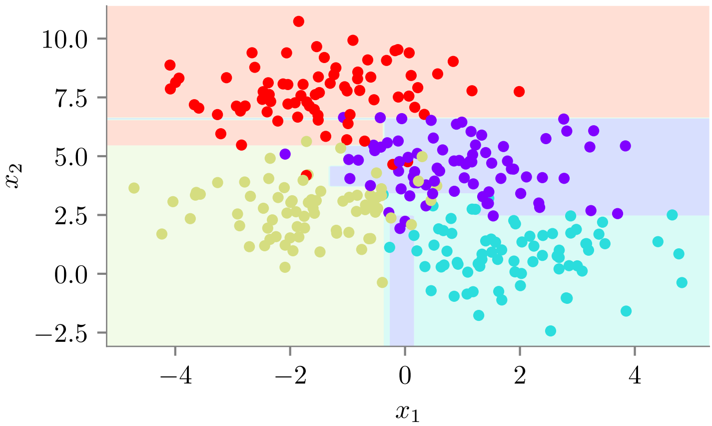
Example (DT of depth 4)

Notebook: decision-tree-real-input-discrete-output.html



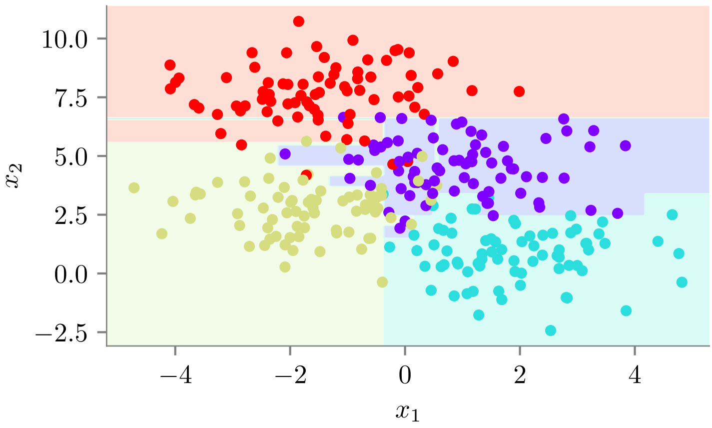
Example (DT of depth 5)

Notebook: decision-tree-real-input-discrete-output.html



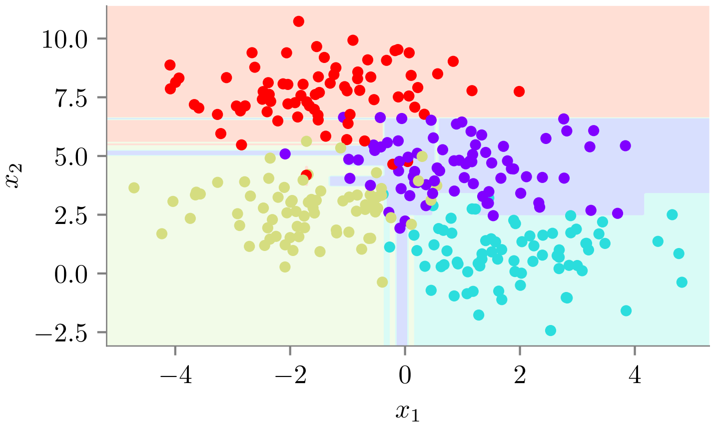
Example (DT of depth 6)

Notebook: decision-tree-real-input-discrete-output.html



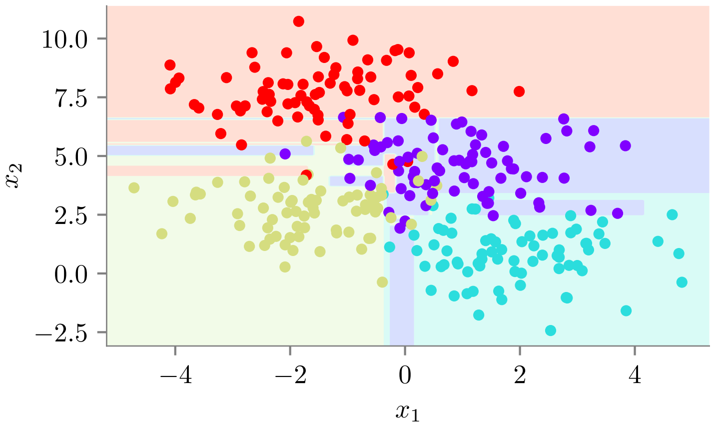
Example (DT of depth 7)

Notebook: decision-tree-real-input-discrete-output.html



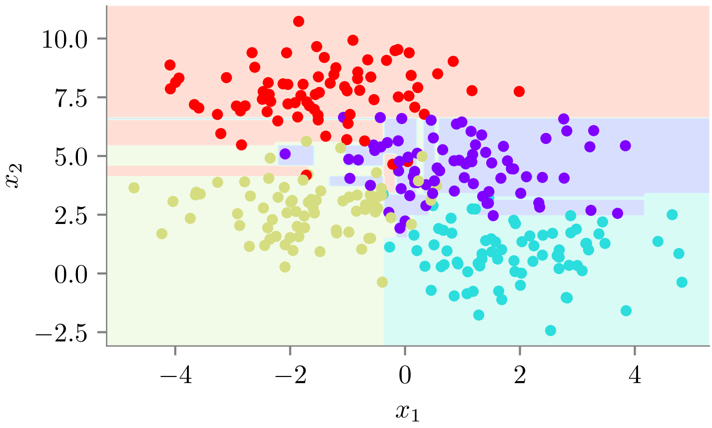
Example (DT of depth 8)

Notebook: decision-tree-real-input-discrete-output.html



Example (DT of depth 9)

Notebook: decision-tree-real-input-discrete-output.html



Pop Quiz #7

Answer this!

When finding splits for continuous features, how do we determine candidate split points?

A) Use all feature values as split points
B) Use midpoints between consecutive sorted feature values
C) Use random values within the feature range
D) Use only the minimum and maximum values

Pop Quiz #7

Answer this!

When finding splits for continuous features, how do we determine candidate split points?

A) Use all feature values as split points

B) Use midpoints between consecutive sorted feature values

C) Use random values within the feature range

D) Use only the minimum and maximum values

Answer: **B) Use midpoints between consecutive sorted feature values** - This ensures we test all meaningful boundaries between different class regions.

Real Input, Real Output

Completing Our Journey: The Fourth Case

Key Points

Our Decision Tree Journey - Final Stop:

- **Discrete Input, Discrete Output**: Categorical splits, entropy

What we learned:

Completing Our Journey: The Fourth Case

Key Points

Our Decision Tree Journey - Final Stop:

- **Discrete Input, Discrete Output**: Categorical splits, entropy
- **Real Input, Discrete Output**: Continuous features, classification (Case 3)

What we learned:

Completing Our Journey: The Fourth Case

Key Points

Our Decision Tree Journey - Final Stop:

- **Discrete Input, Discrete Output**: Categorical splits, entropy
- **Real Input, Discrete Output**: Continuous features, classification (Case 3)
- **Real Input, Real Output**: Continuous features, regression (Case 2 - revisited!)

What we learned:

Completing Our Journey: The Fourth Case

Key Points

Our Decision Tree Journey - Final Stop:

- **Discrete Input, Discrete Output**: Categorical splits, entropy
- **Real Input, Discrete Output**: Continuous features, classification (Case 3)
- **Real Input, Real Output**: Continuous features, regression (Case 2 - revisited!)
- **Real Input, Real Output**: Let's build intuition from Cases 2 & 3!

What we learned:

Completing Our Journey: The Fourth Case

Key Points

Our Decision Tree Journey - Final Stop:

- **Discrete Input, Discrete Output**: Categorical splits, entropy
- **Real Input, Discrete Output**: Continuous features, classification (Case 3)
- **Real Input, Real Output**: Continuous features, regression (Case 2 - revisited!)
- **Real Input, Real Output**: Let's build intuition from Cases 2 & 3!

What we learned:

- From **Case 2**: Use MSE for regression, predict means in leaves

Completing Our Journey: The Fourth Case

Key Points

Our Decision Tree Journey - Final Stop:

- **Discrete Input, Discrete Output**: Categorical splits, entropy
- **Real Input, Discrete Output**: Continuous features, classification (Case 3)
- **Real Input, Real Output**: Continuous features, regression (Case 2 - revisited!)
- **Real Input, Real Output**: Let's build intuition from Cases 2 & 3!

What we learned:

- From **Case 2**: Use MSE for regression, predict means in leaves
- From **Case 3**: Sort data, find midpoint splits, evaluate systematically

Completing Our Journey: The Fourth Case

Key Points

Our Decision Tree Journey - Final Stop:

- **Discrete Input, Discrete Output**: Categorical splits, entropy
- **Real Input, Discrete Output**: Continuous features, classification (Case 3)
- **Real Input, Real Output**: Continuous features, regression (Case 2 - revisited!)
- **Real Input, Real Output**: Let's build intuition from Cases 2 & 3!

What we learned:

- From **Case 2**: Use MSE for regression, predict means in leaves
- From **Case 3**: Sort data, find midpoint splits, evaluate systematically
- **Case 4**: Combine both approaches for regression trees!

The Key Insights from Previous Cases

Example: From Case 3 (Real Input, Discrete Output)

- **Sorting Strategy**: Sort by feature values to find natural boundaries

Example: From Case 2 (Regression Trees)

The Key Insights from Previous Cases

Example: From Case 3 (Real Input, Discrete Output)

- **Sorting Strategy**: Sort by feature values to find natural boundaries
- **Candidate Splits**: Only consider midpoints between different points

Example: From Case 2 (Regression Trees)

The Key Insights from Previous Cases

Example: From Case 3 (Real Input, Discrete Output)

- **Sorting Strategy**: Sort by feature values to find natural boundaries
- **Candidate Splits**: Only consider midpoints between different points
- **Systematic Evaluation**: Test each split, choose best information gain

Example: From Case 2 (Regression Trees)

The Key Insights from Previous Cases

Example: From Case 3 (Real Input, Discrete Output)

- **Sorting Strategy**: Sort by feature values to find natural boundaries
- **Candidate Splits**: Only consider midpoints between different points
- **Systematic Evaluation**: Test each split, choose best information gain

Example: From Case 2 (Regression Trees)

- **MSE Criterion**: Use Mean Squared Error instead of entropy

The Key Insights from Previous Cases

Example: From Case 3 (Real Input, Discrete Output)

- **Sorting Strategy**: Sort by feature values to find natural boundaries
- **Candidate Splits**: Only consider midpoints between different points
- **Systematic Evaluation**: Test each split, choose best information gain

Example: From Case 2 (Regression Trees)

- **MSE Criterion**: Use Mean Squared Error instead of entropy
- **Leaf Predictions**: Predict mean of target values in each region

The Key Insights from Previous Cases

Example: From Case 3 (Real Input, Discrete Output)

- **Sorting Strategy**: Sort by feature values to find natural boundaries
- **Candidate Splits**: Only consider midpoints between different points
- **Systematic Evaluation**: Test each split, choose best information gain

Example: From Case 2 (Regression Trees)

- **MSE Criterion**: Use Mean Squared Error instead of entropy
- **Leaf Predictions**: Predict mean of target values in each region
- **Weighted Loss**: Consider subset sizes in evaluation

The Challenge: Regression with Continuous Features

Important: What Makes This Different?

- **Input**: Continuous features (like Case 3)

Our Strategy:

The Challenge: Regression with Continuous Features

Important: What Makes This Different?

- **Input**: Continuous features (like Case 3)
- **Output**: Continuous targets (like Case 2)

Our Strategy:

The Challenge: Regression with Continuous Features

Important: What Makes This Different?

- **Input**: Continuous features (like Case 3)
- **Output**: Continuous targets (like Case 2)
- **Challenge**: Where to split continuous features for best regression performance?

Our Strategy:

The Challenge: Regression with Continuous Features

Important: What Makes This Different?

- **Input**: Continuous features (like Case 3)
- **Output**: Continuous targets (like Case 2)
- **Challenge**: Where to split continuous features for best regression performance?

Our Strategy:

1. Apply Case 3's sorting and candidate generation approach

The Challenge: Regression with Continuous Features

Important: What Makes This Different?

- **Input**: Continuous features (like Case 3)
- **Output**: Continuous targets (like Case 2)
- **Challenge**: Where to split continuous features for best regression performance?

Our Strategy:

1. Apply Case 3's sorting and candidate generation approach
2. Use Case 2's MSE-based evaluation criteria

The Challenge: Regression with Continuous Features

Important: What Makes This Different?

- **Input**: Continuous features (like Case 3)
- **Output**: Continuous targets (like Case 2)
- **Challenge**: Where to split continuous features for best regression performance?

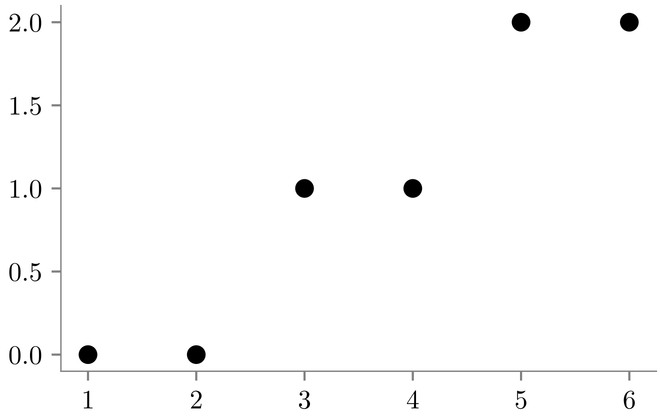
Our Strategy:

1. Apply Case 3's sorting and candidate generation approach
2. Use Case 2's MSE-based evaluation criteria
3. Weight by sample sizes for fair comparison

Example Dataset

Let us consider the regression dataset below:

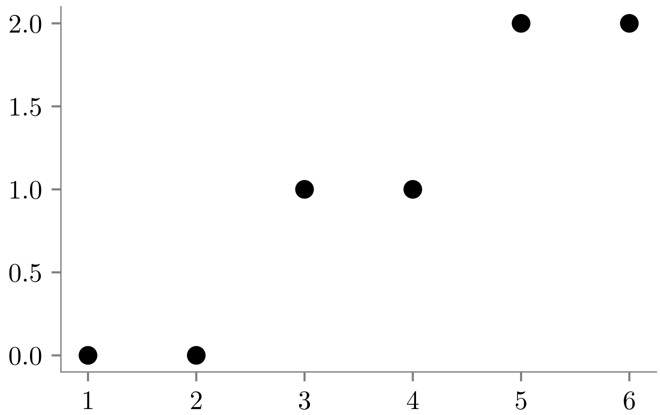
Notebook: decision-tree-real-input-real-output.html



Baseline: Decision Tree with Depth 0

Question: What would be the prediction for decision tree with depth 0 (no splits)?

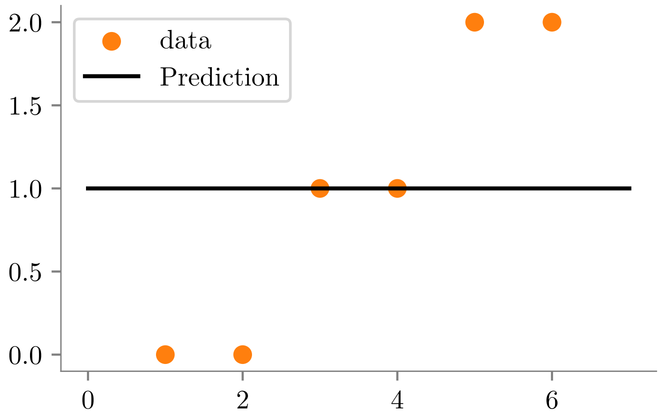
Notebook: decision-tree-real-input-real-output.html



Baseline Performance Visualization

Depth 0 Prediction: Horizontal line = average of all Y values

Notebook: decision-tree-real-input-real-output.html



Finding Split Candidates - Learning from Case 3

Definition: Split Candidate Strategy (from Case 3)

1. **Sort** data points by X values: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

Why midpoints? They ensure we capture all meaningful boundaries where the trend might change.

Finding Split Candidates - Learning from Case 3

Definition: Split Candidate Strategy (from Case 3)

1. **Sort** data points by X values:
   $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
2. **Consider midpoints** between consecutive X values:

$$\text{Split candidates} = \left\{ \frac{x_i + x_{i+1}}{2} \mid i = 1, 2, \ldots, n-1 \right\}$$

Why midpoints? They ensure we capture all meaningful boundaries where the trend might change.

Finding Split Candidates - Learning from Case 3

Definition: Split Candidate Strategy (from Case 3)

1. **Sort** data points by X values:
   $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
2. **Consider midpoints** between consecutive X values:

$$\text{Split candidates} = \left\{ \frac{x_i + x_{i+1}}{2} \mid i = 1, 2, \ldots, n-1 \right\}$$

3. **Evaluate** each split using MSE reduction (not entropy!)

Why midpoints? They ensure we capture all meaningful boundaries where the trend might change.

Objective Function for Regression Trees - Setup

Feature is denoted by X and target by Y.
Let the split be at $X = s$.
Define regions: $R_1 = \{x : x \leq s\}$ and $R_2 = \{x : x > s\}$.
For each region, compute the mean prediction:

$$c_1 = \frac{1}{|R_1|} \sum_{x_i \in R_1} y_i \quad \text{and} \quad c_2 = \frac{1}{|R_2|} \sum_{x_i \in R_2} y_i$$

Objective Function - Weighted MSE

Example: Sample-Weighted MSE

$$\text{Weighted Loss}(s) = \frac{|R_1|}{|R_1| + |R_2|} \cdot \text{MSE}(R_1) + \frac{|R_2|}{|R_1| + |R_2|} \cdot \text{MSE}(R_2)$$

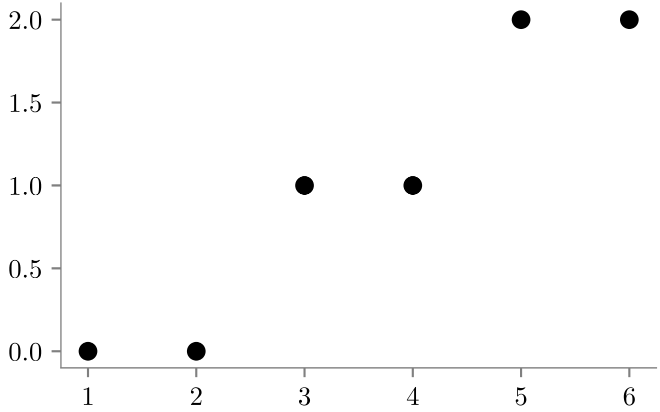
Where: $\text{MSE}(R_i) = \frac{1}{|R_i|} \sum_{x_j \in R_i} (y_j - c_i)^2$

Our objective: $s^* = \arg\min_s \text{Weighted Loss}(s)$

Example: Finding the Best Split

What would be the decision tree with depth 1?

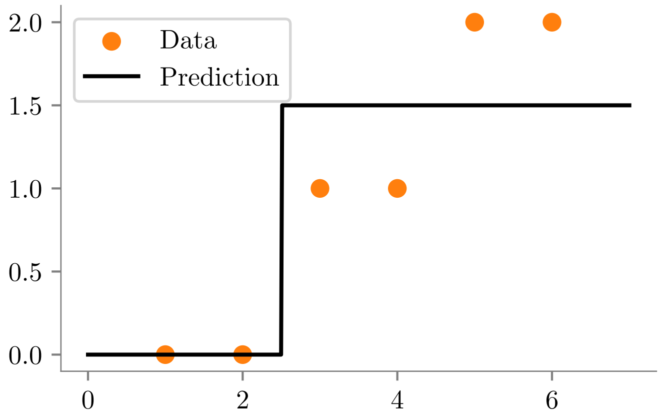
Notebook: decision-tree-real-input-real-output.html



Example 1

Decision tree with depth 1

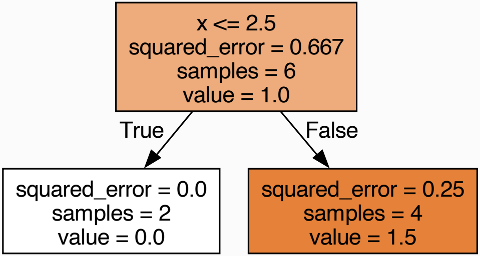
Notebook: decision-tree-real-input-real-output.html



Example 1

The Decision Boundary

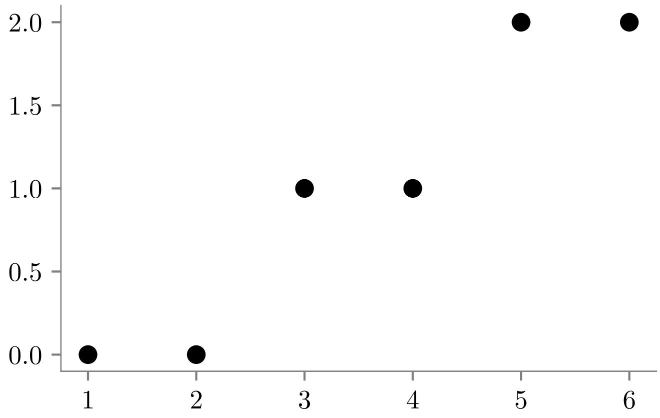
Notebook: decision-tree-real-input-real-output.html



Example 1

What would be the decision tree with depth 2 ?

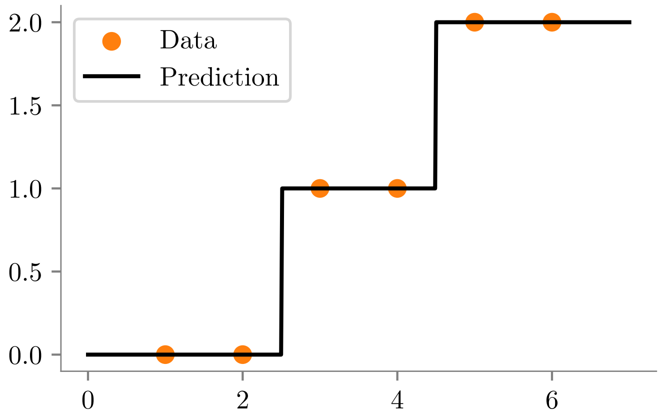
Notebook: decision-tree-real-input-real-output.html



Example 1

Decision tree with depth 2

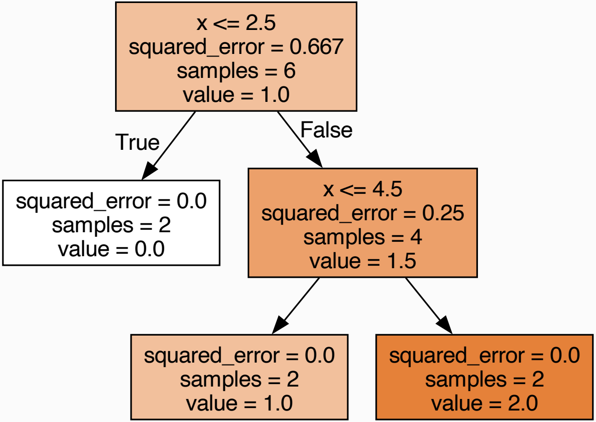
Notebook: decision-tree-real-input-real-output.html



Example 1

The Decision Boundary

Notebook: decision-tree-real-input-real-output.html



Algorithm: Finding the Optimal Split

1. Sort all data points (x_i, y_i) in increasing order of x_i.

Algorithm: Finding the Optimal Split

1. Sort all data points (x_i, y_i) in increasing order of x_i.
2. Evaluate the loss function for all candidate splits:

$$s = \frac{x_i + x_{i+1}}{2} \text{ for } i = 1, 2, \ldots, n - 1$$

3. Select the split s^* that minimizes the loss function.

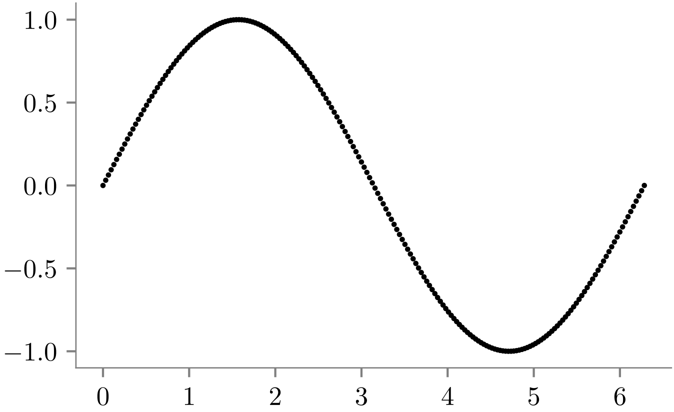
A Question!

Draw a regression tree for $Y = \sin(X)$, $0 \leq X \leq 2\pi$

A Question!

Dataset of Y = sin(X), $0 \leq X \leq 7$ with 10,000 points

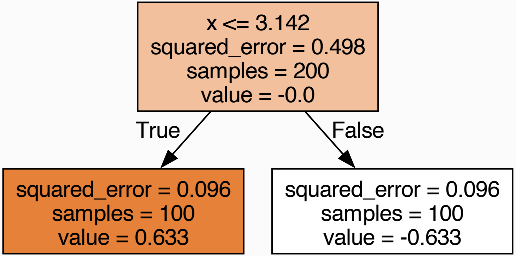
Notebook: decision-tree-real-input-real-output.html



A Question!

Regression tree of depth 1

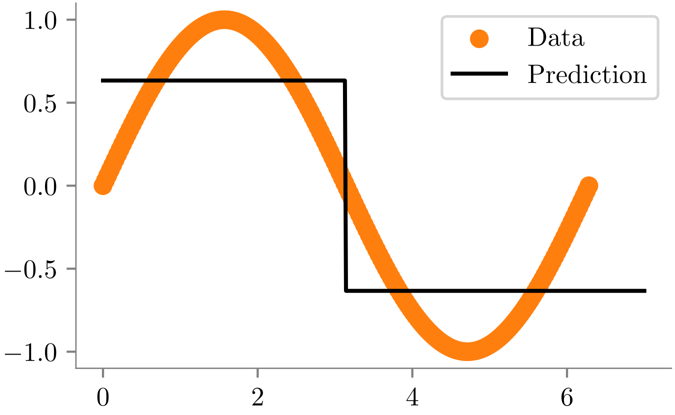
Notebook: decision-tree-real-input-real-output.html



A Question!

Decision Boundary

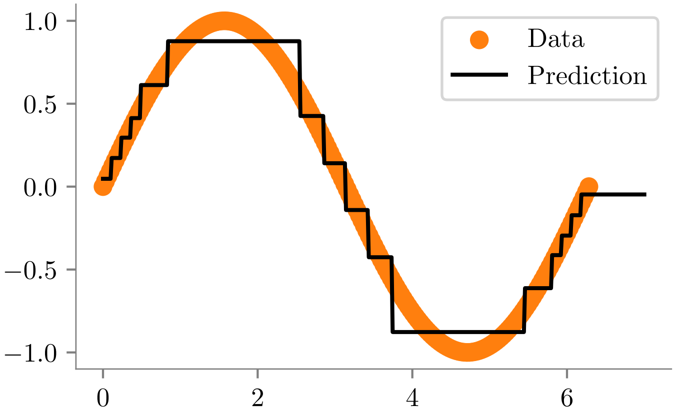
Notebook: decision-tree-real-input-real-output.html



A Question!

Regression tree with no depth limit is too big to fit in a slide. It has of depth 4. The decision boundaries are in figure below.

Notebook: decision-tree-real-input-real-output.html



Pop Quiz #8

Answer this!

What is the prediction function for a regression tree leaf node?

A) The median of target values
B) The mode of target values
C) The mean of target values
D) A linear function

Pop Quiz #8

Answer this!

What is the prediction function for a regression tree leaf node?

A) The median of target values
B) The mode of target values
C) The mean of target values
D) A linear function

Answer: C) The mean of target values

Weighted Entropy

Why Weighted Entropy?

Key Points

- Sometimes points have **different importance**:

Why Weighted Entropy?

Key Points

- Sometimes points have **different importance**:
  - From resampling (e.g., Boosting)

Why Weighted Entropy?

Key Points

- Sometimes points have **different importance**:
  - From resampling (e.g., Boosting)
  - Class imbalance correction

Why Weighted Entropy?

Key Points

- Sometimes points have **different importance**:
  - From resampling (e.g., Boosting)
  - Class imbalance correction
  - Prior knowledge about reliability of examples

Why Weighted Entropy?

Key Points

- Sometimes points have **different importance**:
  - From resampling (e.g., Boosting)
  - Class imbalance correction
  - Prior knowledge about reliability of examples
- Standard entropy assumes equal weight for all points.

Why Weighted Entropy?

Key Points

- Sometimes points have **different importance**:
  - From resampling (e.g., Boosting)
  - Class imbalance correction
  - Prior knowledge about reliability of examples
- Standard entropy assumes equal weight for all points.
- **Weighted entropy** respects point importance.

Weighted Entropy Formula

For dataset S with points (x_j, y_j) and weights w_j:

$$H_w(S) = -\sum_{c \in \mathcal{C}} \underbrace{\frac{\sum_{j \in c} w_j}{\sum_{j \in S} w_j}}_{\text{Weighted } P(c)} \log_2 \left(\frac{\sum_{j \in c} w_j}{\sum_{j \in S} w_j} \right)$$

- w_j: weight of point j (can be fractional)

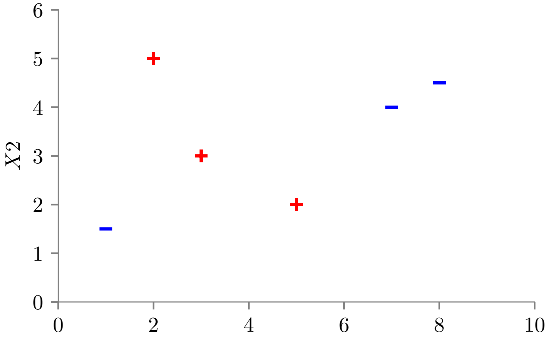
Weighted Entropy Formula

For dataset S with points (x_j, y_j) and weights w_j:

$$H_w(S) = -\sum_{c \in \mathcal{C}} \underbrace{\frac{\sum_{j \in c} w_j}{\sum_{j \in S} w_j}}_{\text{Weighted } P(c)} \log_2 \left(\frac{\sum_{j \in c} w_j}{\sum_{j \in S} w_j} \right)$$

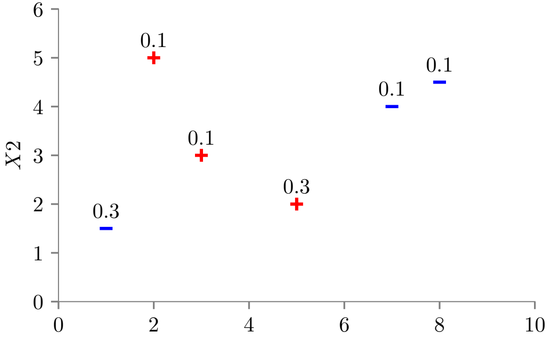
- w_j: weight of point j (can be fractional)
- $\mathcal{C}$: set of classes

Weighted Entropy



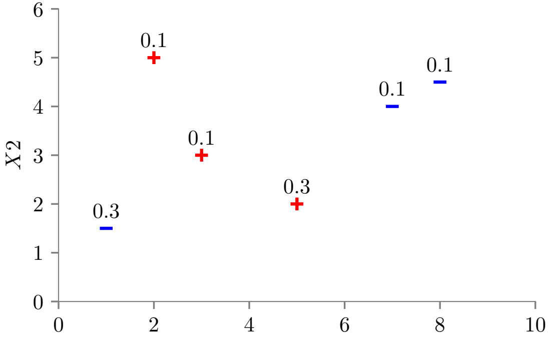
Setup: We start with mixed data - let's see how splitting helps us achieve purity

Weighted Entropy



Original Entropy Calculation:

Computing Original Entropy



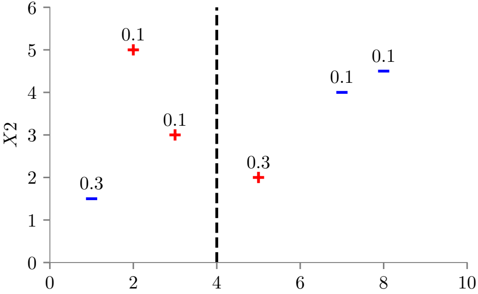
$$\text{Entropy} = -P(+)\log_2 P(+) - P(-)\log_2 P(-)$$

$$P(+) = \frac{0.1 + 0.1 + 0.3}{1} = 0.5, \quad P(-) = \frac{0.3 + 0.1 + 0.1}{1} = 0.5$$

$$\text{Entropy} = E_s = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$

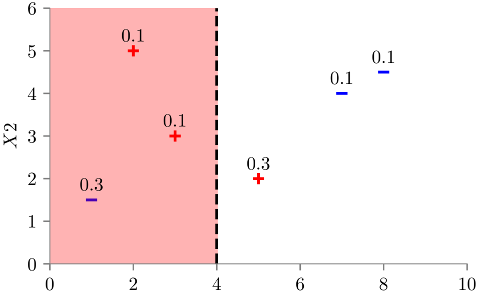
Maximum impurity! Perfect 50-50 split

Choosing a Split



Candidate Split: $X_1 = 4$ (denoted as X_1^*)
Let's see if this vertical line creates purer subsets!

Left Subset: $X_1 \leq X_1^*$



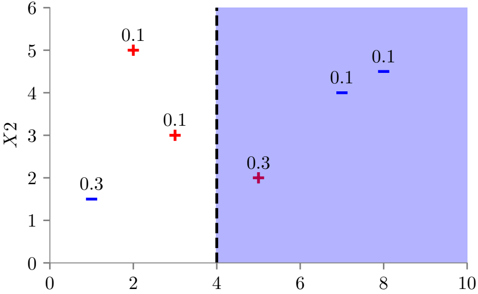
Computing entropy of left subset:

$$P(+) = \frac{0.1 + 0.1}{0.1 + 0.1 + 0.3} = \frac{2}{5} = 0.4$$

$$P(-) = \frac{0.3}{0.5} = \frac{3}{5} = 0.6$$

$$H(\text{left}) = -0.4 \log_2(0.4) - 0.6 \log_2(0.6) \approx 0.971$$

Right Subset: $X_1 > X_1^*$



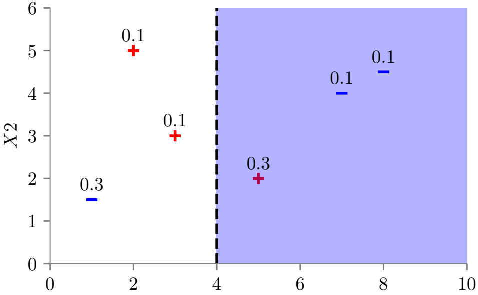
Computing entropy of right subset:

$$P(+) = \frac{0.3}{0.3 + 0.2} = \frac{3}{5} = 0.6$$

$$P(-) = \frac{0.2}{0.5} = \frac{2}{5} = 0.4$$

$$H(\text{right}) = -0.6 \log_2(0.6) - 0.4 \log_2(0.4) \approx 0.971$$

Computing Weighted Entropy



Example: Weighted Entropy Calculation

$$\text{Weighted Entropy} = \frac{|S_{\text{left}}|}{|S|} \cdot H(\text{left}) + \frac{|S_{\text{right}}|}{|S|} \cdot H(\text{right})$$

$$= \frac{0.5}{1.0} \cdot 0.971 + \frac{0.5}{1.0} \cdot 0.971$$

$$= 0.5 \times 0.971 + 0.5 \times 0.971 = 0.971$$

Information Gain Calculation

Key Points

Information Gain = Original Entropy - Weighted Entropy

$$\text{IG}(X_1 = X_1^*) = E_S - \text{Weighted Entropy}$$

$$= 1.0 - 0.971 = 0.029$$

Interpretation: Small gain means this split doesn't help much in creating purer subsets. We should try other splits!

Key Takeaways

- Weighted entropy generalizes normal entropy.

Key Takeaways

- Weighted entropy generalizes normal entropy.
- Handles fractional and non-uniform point importance.

Key Takeaways

- Weighted entropy generalizes normal entropy.
- Handles fractional and non-uniform point importance.
- Essential for boosting and class imbalance handling.

Key Takeaways

- Weighted entropy generalizes normal entropy.
- Handles fractional and non-uniform point importance.
- Essential for boosting and class imbalance handling.
- Always normalize weights before computing probabilities.

Pruning and Overfitting

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example
- **But**: Poor generalization to new data

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example
- **But**: Poor generalization to new data
- **Symptoms**:

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example
- **But**: Poor generalization to new data
- **Symptoms**:
  - High training accuracy, low test accuracy

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example
- **But**: Poor generalization to new data
- **Symptoms**:
  - High training accuracy, low test accuracy
  - Very deep trees with many leaves

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example
- **But**: Poor generalization to new data
- **Symptoms**:
  - High training accuracy, low test accuracy
  - Very deep trees with many leaves
  - Rules that are too specific to training data

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example
- **But**: Poor generalization to new data
- **Symptoms**:
  - High training accuracy, low test accuracy
  - Very deep trees with many leaves
  - Rules that are too specific to training data
- **Solution**: Pruning to control model complexity

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth = 5)

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth = 5)
- **Minimum samples per split**: Don't split if node has < N samples

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth $= 5$)
- **Minimum samples per split**: Don't split if node has $<$ N samples
- **Minimum samples per leaf**: Ensure each leaf has $\geq$ M samples

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth = 5)
- **Minimum samples per split**: Don't split if node has $<$ N samples
- **Minimum samples per leaf**: Ensure each leaf has $\geq$ M samples
- **Maximum features**: Consider only subset of features at each split

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth = 5)
- **Minimum samples per split**: Don't split if node has $<$ N samples
- **Minimum samples per leaf**: Ensure each leaf has $\geq$ M samples
- **Maximum features**: Consider only subset of features at each split
- **Minimum impurity decrease**: Only split if improvement $>$ threshold

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth = 5)
- **Minimum samples per split**: Don't split if node has $<$ N samples
- **Minimum samples per leaf**: Ensure each leaf has $\geq$ M samples
- **Maximum features**: Consider only subset of features at each split
- **Minimum impurity decrease**: Only split if improvement $>$ threshold

Advantages: Simple, computationally efficient

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth = 5)
- **Minimum samples per split**: Don't split if node has $<$ N samples
- **Minimum samples per leaf**: Ensure each leaf has $\geq$ M samples
- **Maximum features**: Consider only subset of features at each split
- **Minimum impurity decrease**: Only split if improvement $>$ threshold

Advantages: Simple, computationally efficient
Disadvantages: May stop too early, miss good splits later

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data
  2. Use validation set to evaluate subtree performance

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data
  2. Use validation set to evaluate subtree performance
  3. Remove branches that don't improve validation accuracy

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data
  2. Use validation set to evaluate subtree performance
  3. Remove branches that don't improve validation accuracy
  4. Repeat until no beneficial removals remain

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data
  2. Use validation set to evaluate subtree performance
  3. Remove branches that don't improve validation accuracy
  4. Repeat until no beneficial removals remain
- **Cost Complexity Pruning**: Minimize Error $+ \alpha \times$ Tree Size

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data
  2. Use validation set to evaluate subtree performance
  3. Remove branches that don't improve validation accuracy
  4. Repeat until no beneficial removals remain
- **Cost Complexity Pruning**: Minimize Error $+ \alpha \times$ Tree Size
- **Advantages**: More thorough, can recover from early stopping mistakes

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data
  2. Use validation set to evaluate subtree performance
  3. Remove branches that don't improve validation accuracy
  4. Repeat until no beneficial removals remain
- **Cost Complexity Pruning**: Minimize Error $+ \alpha \times$ Tree Size
- **Advantages**: More thorough, can recover from early stopping mistakes
- **Disadvantages**: More computationally expensive

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Misclassification error on validation set

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Misclassification error on validation set
  - $|T|$: Number of terminal nodes (tree size)

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Misclassification error on validation set
  - $|T|$: Number of terminal nodes (tree size)
  - α: Complexity parameter (penalty for larger trees)

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Misclassification error on validation set
  - $|T|$: Number of terminal nodes (tree size)
  - α: Complexity parameter (penalty for larger trees)
- **Process**:

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
    - $R(T)$: Misclassification error on validation set
    - $|T|$: Number of terminal nodes (tree size)
    - α: Complexity parameter (penalty for larger trees)
- **Process**:
    1. Start with full tree ($\alpha = 0$)

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Misclassification error on validation set
  - $|T|$: Number of terminal nodes (tree size)
  - α: Complexity parameter (penalty for larger trees)
- **Process**:
  1. Start with full tree ($\alpha = 0$)
  2. Gradually increase α

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Misclassification error on validation set
  - $|T|$: Number of terminal nodes (tree size)
  - α: Complexity parameter (penalty for larger trees)
- **Process**:
  1. Start with full tree ($\alpha = 0$)
  2. Gradually increase α
  3. At each α, prune branches that increase cost

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Misclassification error on validation set
  - $|T|$: Number of terminal nodes (tree size)
  - α: Complexity parameter (penalty for larger trees)
- **Process**:
  1. Start with full tree ($\alpha = 0$)
  2. Gradually increase α
  3. At each α, prune branches that increase cost
  4. Select α with best cross-validation performance

Bias-Variance Trade-off in Trees

- **Unpruned trees**:

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting
- **Heavily pruned trees**:

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting
- **Heavily pruned trees**:
  - High bias (may miss important patterns)

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting
- **Heavily pruned trees**:
  - High bias (may miss important patterns)
  - Low variance (more stable predictions)

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting
- **Heavily pruned trees**:
  - High bias (may miss important patterns)
  - Low variance (more stable predictions)
  - Risk of underfitting

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting
- **Heavily pruned trees**:
  - High bias (may miss important patterns)
  - Low variance (more stable predictions)
  - Risk of underfitting
- **Optimal pruning**: Balances bias and variance

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting
- **Heavily pruned trees**:
  - High bias (may miss important patterns)
  - Low variance (more stable predictions)
  - Risk of underfitting
- **Optimal pruning**: Balances bias and variance
- **Cross-validation**: Essential for finding this balance

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity
- **Common parameters** (sklearn):

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity
- **Common parameters** (sklearn):
  - `max_depth`: Start with 3-10

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity
- **Common parameters** (sklearn):
  - `max_depth`: Start with 3-10
  - `min_samples_split`: Try 10-100

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity
- **Common parameters** (sklearn):
  - `max_depth`: Start with 3-10
  - `min_samples_split`: Try 10-100
  - `min_samples_leaf`: Try 5-50

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity
- **Common parameters** (sklearn):
  - `max_depth`: Start with 3-10
  - `min_samples_split`: Try 10-100
  - `min_samples_leaf`: Try 5-50
  - `ccp_alpha`: Use for cost complexity pruning

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity
- **Common parameters** (sklearn):
  - `max_depth`: Start with 3-10
  - `min_samples_split`: Try 10-100
  - `min_samples_leaf`: Try 5-50
  - `ccp_alpha`: Use for cost complexity pruning
- **Domain knowledge**: Consider interpretability requirements

Summary and Key Takeaways

Summary and Key Takeaways

- Interpretability an important goal

Summary and Key Takeaways

- Interpretability an important goal
- Decision trees: well known interpretable models

Summary and Key Takeaways

- Interpretability an important goal
- Decision trees: well known interpretable models
- Learning optimal tree is hard

Summary and Key Takeaways

- Interpretability an important goal
- Decision trees: well known interpretable models
- Learning optimal tree is hard
- Greedy approach:

Summary and Key Takeaways

- Interpretability an important goal
- Decision trees: well known interpretable models
- Learning optimal tree is hard
- Greedy approach:
- Recursively split to maximize "performance gain"

Summary and Key Takeaways

- Interpretability an important goal
- Decision trees: well known interpretable models
- Learning optimal tree is hard
- Greedy approach:
- Recursively split to maximize “performance gain”
- Issues:

Summary and Key Takeaways

- Interpretability an important goal
- Decision trees: well known interpretable models
- Learning optimal tree is hard
- Greedy approach:
- Recursively split to maximize "performance gain"
- Issues:
    - Can overfit easily!

Summary and Key Takeaways

- Interpretability an important goal
- Decision trees: well known interpretable models
- Learning optimal tree is hard
- Greedy approach:
- Recursively split to maximize “performance gain”
- Issues:
  - Can overfit easily!
  - Empirically not as powerful as other methods